Application of Geometric Modeling to Mechanical Assembly

by
SANJAY T. TUPE

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

February, 2001

Application of Geometric Modeling to Mechanical Assembly

A thesis submitted
in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Sanjay T. Tupe

to the

Department of Mechanical Engineering
Indian Institute of Technology
Kanpur-208016, India

February, 2001

CERTIFICATE

It is certified that the work contained in the thesis entitled ***Application of Geometric Modeling to Mechanical Assembly*** by **Sanjay T. Tupe** *(Roll No:9910543)* has been carried out under my supervision and the work has not been submitted elsewhere for a degree.

B Sahay

Prof. B. Sahay
Department of Mechanical Engineering,
Indian Institute of Technology, Kanpur

Feb, 2001

Acknowledgment

I would like to express my profound, most sincere appreciation and special thanks to my guide Prof. B. Sahay for his careful guidance and continuous encouragement during the entire period of this study.

I am extremely grateful to Prof. S. G. Dhande and the staff of the CAD-P laboratory for providing me all sort of facilities available in their laboratory in support of this thesis work.

I am grateful to all members of the faculty, who imparted great knowledge on me by their lectures.

Many thanks are due to the classmates for their smiles and friendship making the life at IIT, Kanpur memorable.

Last, but not least, I wish to express my wholehearted gratitude to my beloved parents for their blessings. These humble accomplishments are only a small reflection of their tireless and uncompromising efforts. This humble piece of work is dedicated to them.

Name of Student : Sanjay T. Tupe **Roll No. :** 9910543
Degree for which submitted : M. Tech. **Department :** Mechanical Engineering

Thesis title : *Application of Geometric Modeling to Mechanical Assembly.*

Name of Thesis Supervisor :

Dr. B. Sahay

Month and Year of Thesis Submission: February, 2001

Abstract

The primary focus in geometric modeling and computer-aided design has been on the analysis of individual components. However in practical engineering applications, there is often the need to examine the performance of an assembly of components rather than the individual components. A very welcome development would be the introduction of a software package which would allow a designer to create individual components, assemble them and then perform the necessary analysis on them.

Conventionally, if the designer wants to inspect the dynamic or kinematic performance of an assembly, he has to generate a model of the assembly for the analysis from the assembly drawings. Also, he has to regenerate the model whenever any component is modified. This additional step tends to lead the designer to either consider fewer design alternatives or to skip the analysis.

This work provides a simple and user friendly assembly modeling system for all mechanical parts. This is achieved using two approaches, one is Homogeneous Transformation Matrix approach and the other is Mating Conditions approach. These two approaches were successfully tested for performing different mechanical assemblies with different mating conditions. Although both the approaches require the user to input the necessary geometric information, each of them has its own advantages and limitations. Although there are many assembly sequences possible for the cases we have solved, the most convenient assembly sequence is being used. The assembly sequences used are manual ones.

In this work, OpenGL (Open Graphics Library) is used for rendering different components of an assembly. The algorithm for assembly procedure has been written in "C" language which has good interface with OpenGL. The representation scheme being used is Boundary representation (B-rep) as the mating conditions are related to the faces, edges and vertices of the assembled parts.

Although this work is complete up to the assembly modeling stage, one can extend it for performing assembly analysis such as interference checking, mass property calculations and finite element analysis. There is also a scope for extending this work for generating automatic assembly sequences from the mating features of the various components of the assembly.

Contents

List of Figures

Chapter 1

Introduction

The primary focus in geometric modeling and computer-aided design has been on the design and analysis of individual components. However in practical engineering applications, there is often the need to examine the performance of an assembly of components rather than the individual components. A very welcome development would be the introduction of a software package which would allow a designer to create individual components, assemble them and then perform the necessary analysis on them.

Conventionally, if the designer wants to inspect the dynamic or kinematic performance of an assembly, he has to generate a model of the assembly for the analysis from the assembly drawings. Also, he has to regenerate the model whenever any component is modified. This additional step tends to lead the designer to either consider fewer design alternatives or to skip the analysis.

The objective of this study, although it is complete only up to the assembly modeling stage, is to create an integrated procedure in which the designer can create individual components, assemble them into an assembly and perform the dynamic or kinematic analysis at any time during the course of design. With this procedure there would be no extra effort required by the designer for the analysis. Thus, the analysis can be performed for all the alternative designs.

1.1 Literature Survey

An assembly of components can be represented by the description of its individual components in the assembly and the relationship between the components in the assembly. The representation schemes of individual components have been well established. In this work, Boundary representation (B-rep) is chosen over constructive solid geometry (CSG) representation for interactive assignment of mating conditions between components in an assembly. However, the assembly data structure to represent the relationship between components has not been well established. The inherent problem in most assembly data structures is that they lack the ability of interactive assignment in building or developing the assembly. For example

in the assembly data structures introduced by Wesley *et al*[5], the location and orientation of each component in the assembly is specified by a 4×4 homogeneous transformation matrix. The provision of the transformation matrices for each respective component is done in a non-interactive manner. Specifying each transformation matrix for each component can be quite awkward and tends to be error prone, and therefore is better to avoid.

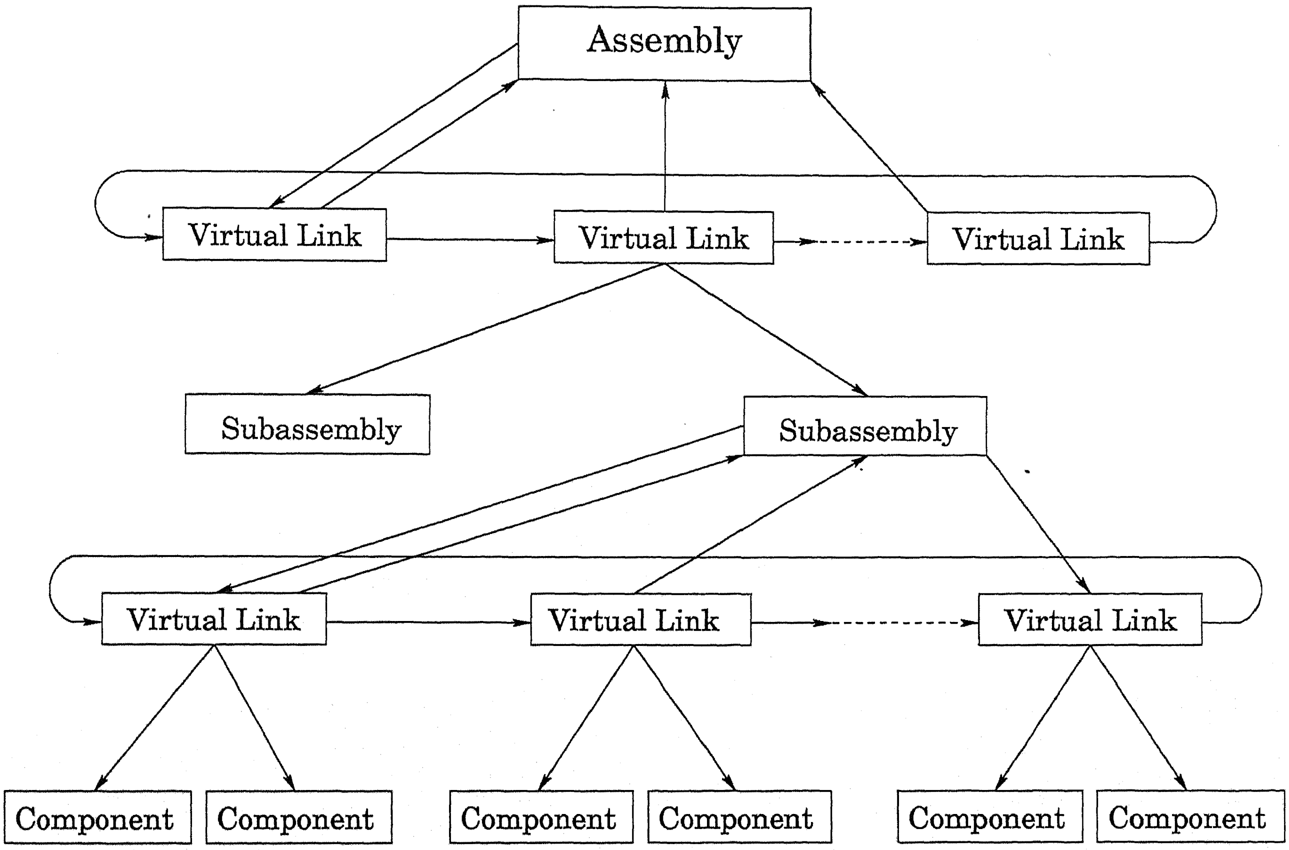

Figure 1.1: Tree structure for assembly

The data structure developed by Lee[3] (Figure 1.1) overcomes the problem of specifying transformation matrices directly by using more basic information between individual components in an assembly through mating conditions. He introduced the concept of a "virtual link" to group all the mating conditions between a pair of mating components into one entity (virtual link), Thus an assembly of components can be represented in a hierarchical tree. In this approach, the top node of the tree is the assembly which is composed of sub-assemblies interrelated by the virtual links. Tracing down the sub-assemblies, other sub-assemblies may be encountered that are related by virtual links which further branch down to the terminal nodes of the tree structure which are the individual components, i.e. if the component is used more than once in the assembly then the concept of "instance" may be used. This allows many identical components or instances to have only one component data structure and the virtual links point to the instances of the component rather than to several identical components.

Since Lee[3] provided the mating feature information, the transformation matrix for each component need not be assigned because it may be derived from the mating feature information carried by virtual links. An added benefit using mating condition information instead of providing the transformation matrices is the determination of whether or not the components in an assembly can be assembled. if it turns out that some components can not be physically be assembled, the transformation matrices satisfying all the mating conditions do not exist and the computation routine for transformation matrices will diverge. In this way only possible assembly can be stored in the database.

In this assembly data structure, the mating conditions are specified instead of the transformation matrices, which require the calculation of transformation matrices from the given mating conditions. Lee and Andrews[2] proposed an approach for deriving the 12 variables (nine rotational and three translational elements) of the homogeneous matrix shown below by using the spatial relationships defined by the mating conditions.

$$T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore to assemble N components, $N-1$ transformation matrices need to be supplied (one component is fixed as the reference component). This indicates that $12 \times (N-1)$ variables must be solved simultaneously. The against and the fits conditions along with the transformation matrix properties provide the constraint equations necessary to solve the $12 \times (N-1)$ variables.

In this method, it turns out that the number of equations is always equal to or greater than the number of variables produced. Therefore, a method must be used which accounts for the number of redundant equations realized. The authors[2] incorporate an algorithm developed by Light which searches for groups of equations which exhibit a linear dependency. By identifying the redundant equations with Light's Routine, the set of equations can now be reduced to a number of equations which equal the number of variables. The linearly independent equations can now be solved via the Newton-Raphson iteration method. The authors[2] point out that the Newton-Raphson iteration can fail due to the non-uniqueness of the non-linear set of equations. They[2] have also noted that the initial guess on the transformation matrix elements can dictate the success of convergence to a correct solution.

Since there are more equations than variables, a suggested alternate solution method to bypass Light's routine in searching for redundant equations is the Least Squares Technique. Although it was suggested, they[2] did not use the least square algorithm.

1.2 General Approach

The general approach used to derive the assembly model and develop the joint information necessary for further analysis is outlined in four steps.

- **Step 1: Represent individual components in a database**
  Any existing Boundary representation scheme can be used to represent each component.

- **Step 2: Specify the relationship between the components in an assembly**
  This involves the problem of how to represent an assembly, i.e. how to store relationship between each component in an assembly.

- **Step 3: Find the location and orientation of their assembled positions**
  To accomplish this step, a computational scheme is employed that uses the two approaches.

- **Step 4: Perform the dynamic and kinematic analysis on the created assembly**
  Though not incorporated in this work, the joint information for this analysis can be derived directly from the assembly model.

1.3 Objective of the Present Work

It is the common practice that a designer creates the model of a mechanical component on a CAD system and the commands for a numerical controlled system or machine are directly generated to manufacture the component. By analogy, it would be very productive if an assembly procedure were generated directly as designer creates a model of an assembly of components. As mentioned earlier, if the designer wants to inspect the dynamic or kinematic performance of an assembly, he has to generate a model of the assembly for the analysis from the assembly drawings. Also he has to regenerate the model whenever any component is modified. This additional step tends to lead the designer either to consider fewer design alternatives or to skip the analysis.

Therefore, the main objectives of this work are:

- To generate an integrated procedure by which the designer can create individual components, assemble them into an assembly and perform the dynamic and kinematic analysis.

- To develop a method in which an assembly procedure is generated automatically from a description of an assembly provided by the designer.

Chapter 2

Review of Mechanical Assembly

2.1 Definition of Assembly

Assembly sometimes designates the process and sometimes the product. Therefore, by process it can be defined as putting various parts together to create an end product and by product it may be defined as a collection of independent parts put together to perform some function.

Assembly includes both reversible fastening processes (screwing, bolting...) and irreversible ones (riveting, welding, glueing).

2.2 Significance of Assembly

In most engineering design, the product of interest is a combination of parts formed into an assembly. When the product is designed, consideration is generally given to the ease of manufacturing its individual parts and how the final product would look. Little attention is usually given to those aspects of design that will facilitate assembly of parts and great reliance is often placed on assembly or production engineers to solve any assembly-related problems. This approach worked well in the past because all the mechanical operations were performed manually, labor was inexpensive and because the products were not complex. As the products are becoming more complex and the labor is becoming more expensive, the demand to pay attention to the assembly process during the design phase of a product is becoming increasingly high. The most obvious way to facilitate the assembly process at the design phase is to simplify the product by reducing the number of different parts to a minimum.

For automatic assembly, each product should have a base part (or a host part) on which the assembly can be built. This base part must be designed with features that make it suitable for quick and accurate location on the assembly line or work carrier.

Apart from the assembly considerations at the design phase, which are discussed above, modeling and representation of assemblies are all relevant issues to geometric modeling and to the CAD/CAM technology. Parts and/or sub-assemblies of a given product can be

modeled separately, most often by the different members of the design team. Instances of these parts can then be merged into the base part or the host to generate assembly.

2.3 Assembly Modeling

An assembly is a collection of independent parts. It is important to understand the nature and the structure of dependencies between parts in an assembly to be able to model the assembly properly. In order to determine, for example, whether a part can be moved and which other parts will move with it, the assembly model must include the spatial position and hierarchical relationship among the parts and assembly or attachment relationships (or mating conditions) between the parts. The modeling representation of hierarchical relationships and mating conditions are what distinguishes between modeling individual parts and assemblies, and consequently between geometric modelers and assembly modelers.

Most of the existing modeling packages offered by today's CAD/CAM systems that are in use in practice can be classified as geometric modelers. The data structure is designed to store and manipulate geometric data of individual parts only. Assembly modelers can be thought of as more advanced geometric modelers where the data structure is extended to allow representation and manipulation of hierarchical relationships and mating conditions. Fig.(2.1) shows how an assembly model can be created. The geometric modeler acts as a preprocessor to the assembly modeler. Designers first create all the shape information (both geometry and topology) of the individual parts. They can also analyze the parts separately. Part analysis may include mass property calculation and finite element analysis. Once the part design is complete, designers can utilize the assembly modelers to create the assembly and analyze it. Creating the assembly from its parts requires specifying the mating and spatial relationships between the parts. Assembly analysis may include interference checking, mass property calculation, kinematic and dynamic analysis, and finite element analysis. The link between the geometric modelers and assembly modelers is established such that designers need only to modify individual parts for design modification by using the geometric modeler, and the assembly model is updated automatically.

There are three steps that are necessary for assembly modeling:

- Modeling of individual parts.
- Specifying the hierarchical relationship between parts in the assembly.
- Specifying the mating conditions between parts or specifying the locations and orientations of the parts in their assembled positions with reference to the host.

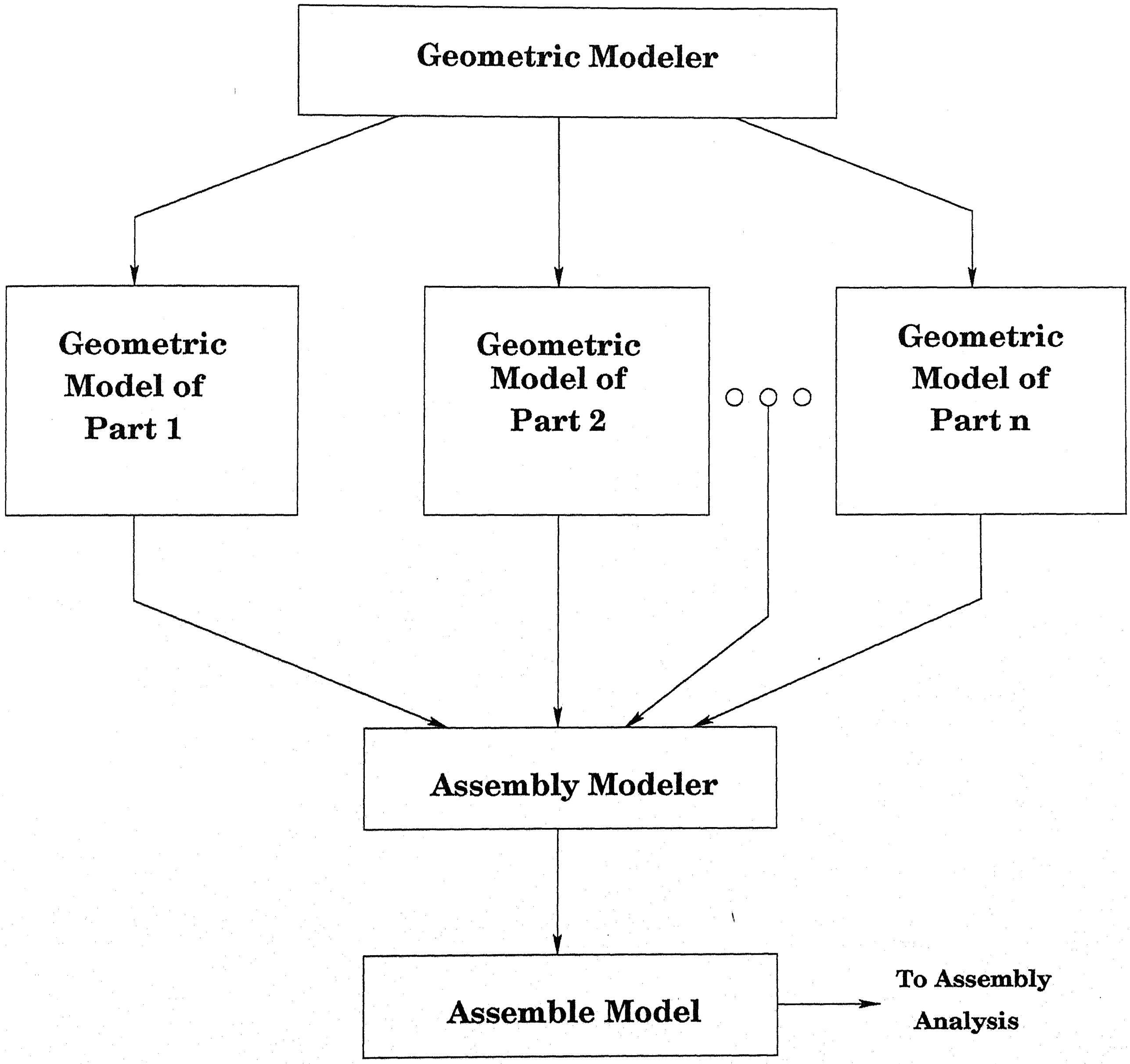

Figure 2.1: Generation of an Assembly model

2.3.1 Parts Modeling and Representation

This is the first step in creating an assembly model. Individual parts can be created using a geometric modeler with the proper representation scheme. Solid modeling, specifically Boundary Representation, is the appropriate scheme because the mating conditions are related to the faces, edges and vertices of the assembled parts. In addition to the shape information, a part database can store assembly attributes such as the parts material type and properties, mass and inertial properties, frictional properties of the faces and others.

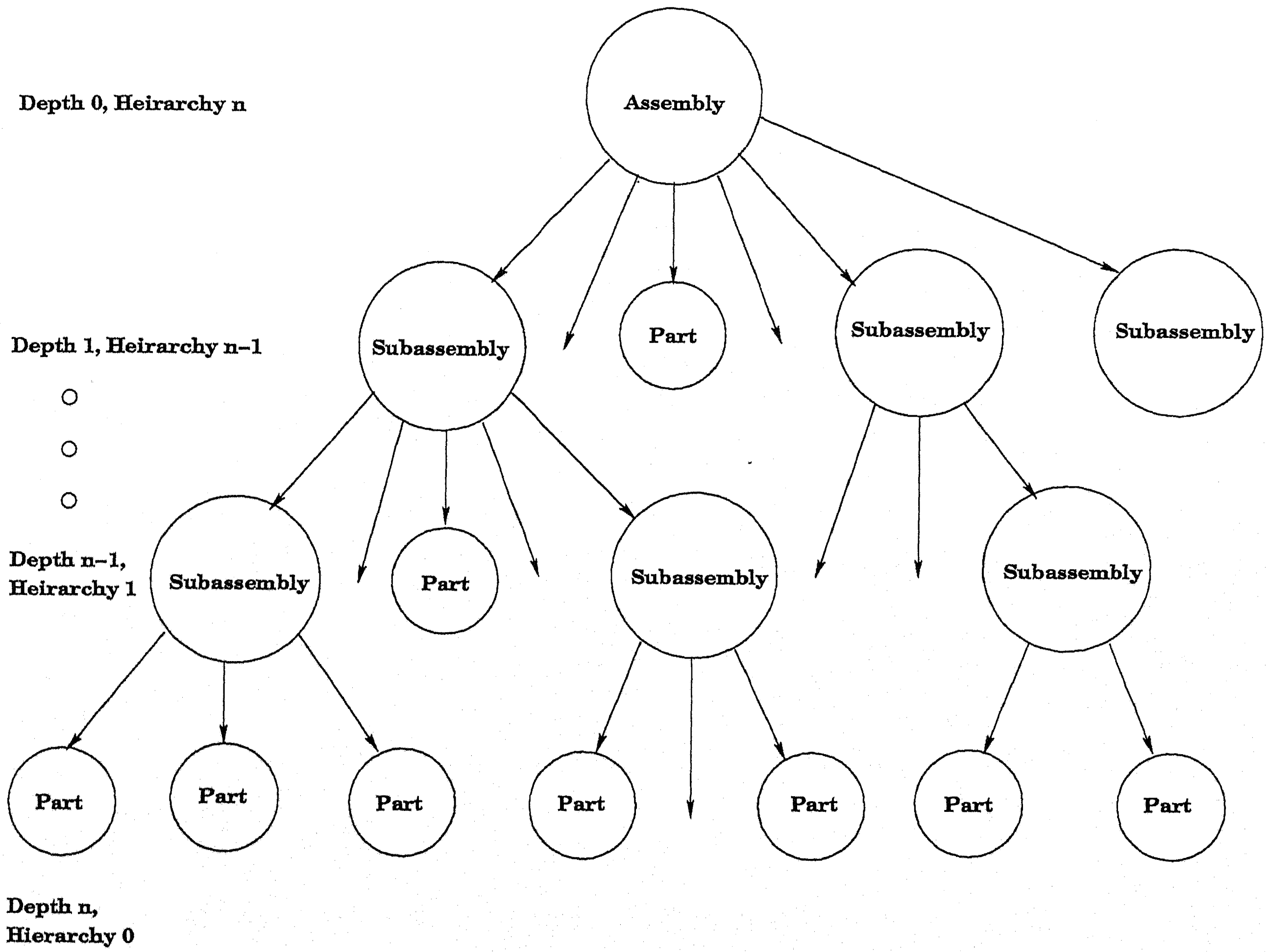

Figure 2.2: Assembly Tree

2.3.2 Hierarchical Relationships

The most natural way to represent the hierarchical relationships between the various parts of an assembly is an assembly tree as shown in Fig.(2.2). An assembly is divided into several assemblies at different levels. If there are n parts in the assembly then each sub-assembly at depth $(n-1)$ is composed of various parts. The leaves of the tree represent individual parts, its nodes represent sub-assemblies and its root represent assembly itself. The

assembly is located at the top of the tree at depth 0 or at the highest hierarchy n of the assembly sequence.

Fig.(2.3) shows a Cardan Universal Joint Assembly. The joint consist of four main elements: Yokes, a block, pins and bushes. Here, for the purpose of assembly the block is held stationary or assumed as the base (host) part. The other parts are located with respect to the base part and then assembled together. The assembly tree for this joint is shown in Fig.(2.4). The tree represents an assembly sequence by which the joint can be produced. The assembly tree is not unique as it is possible to generate other valid assembly sequences.

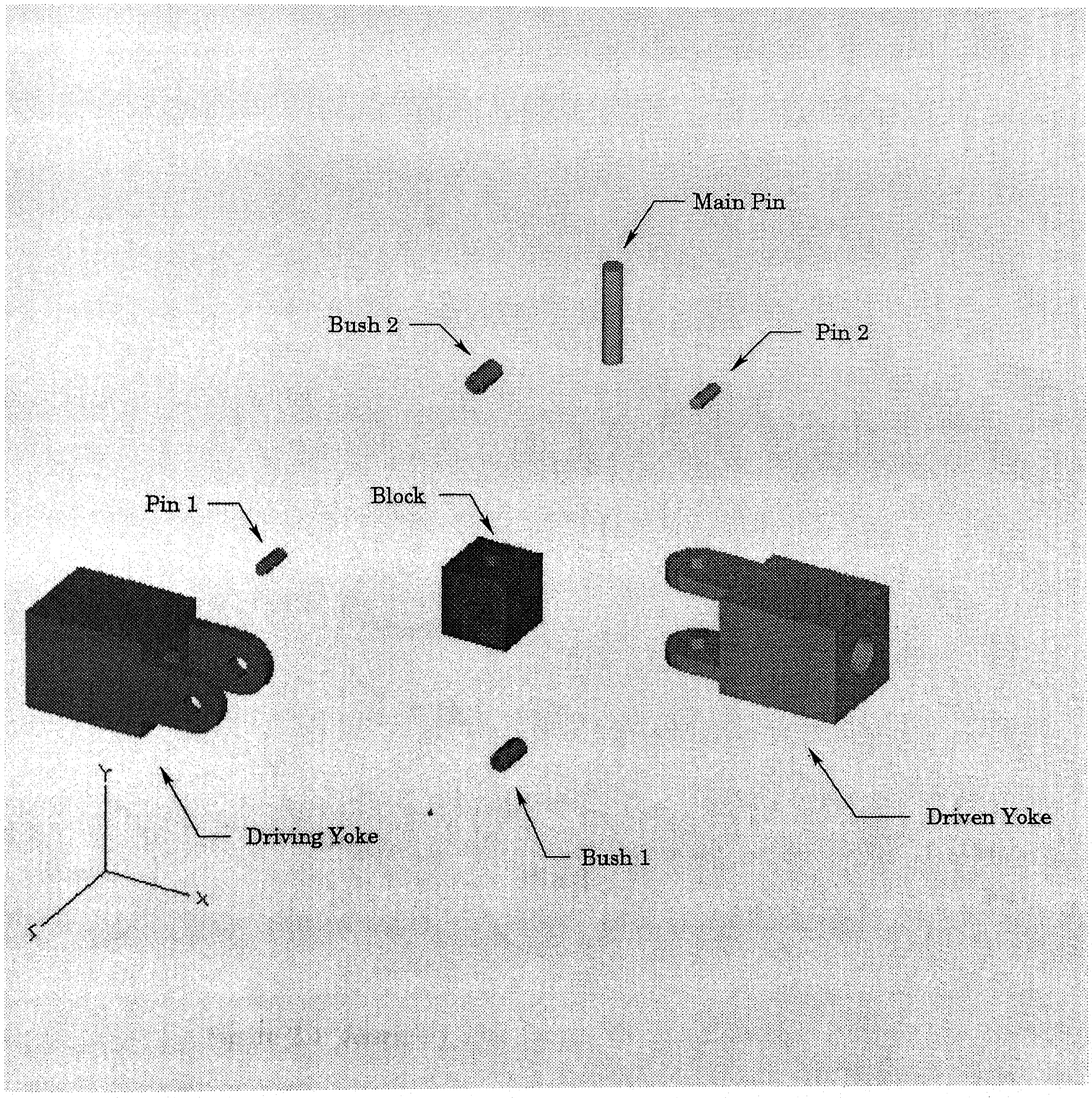

Figure 2.3: Cardan Universal Joint Assembly

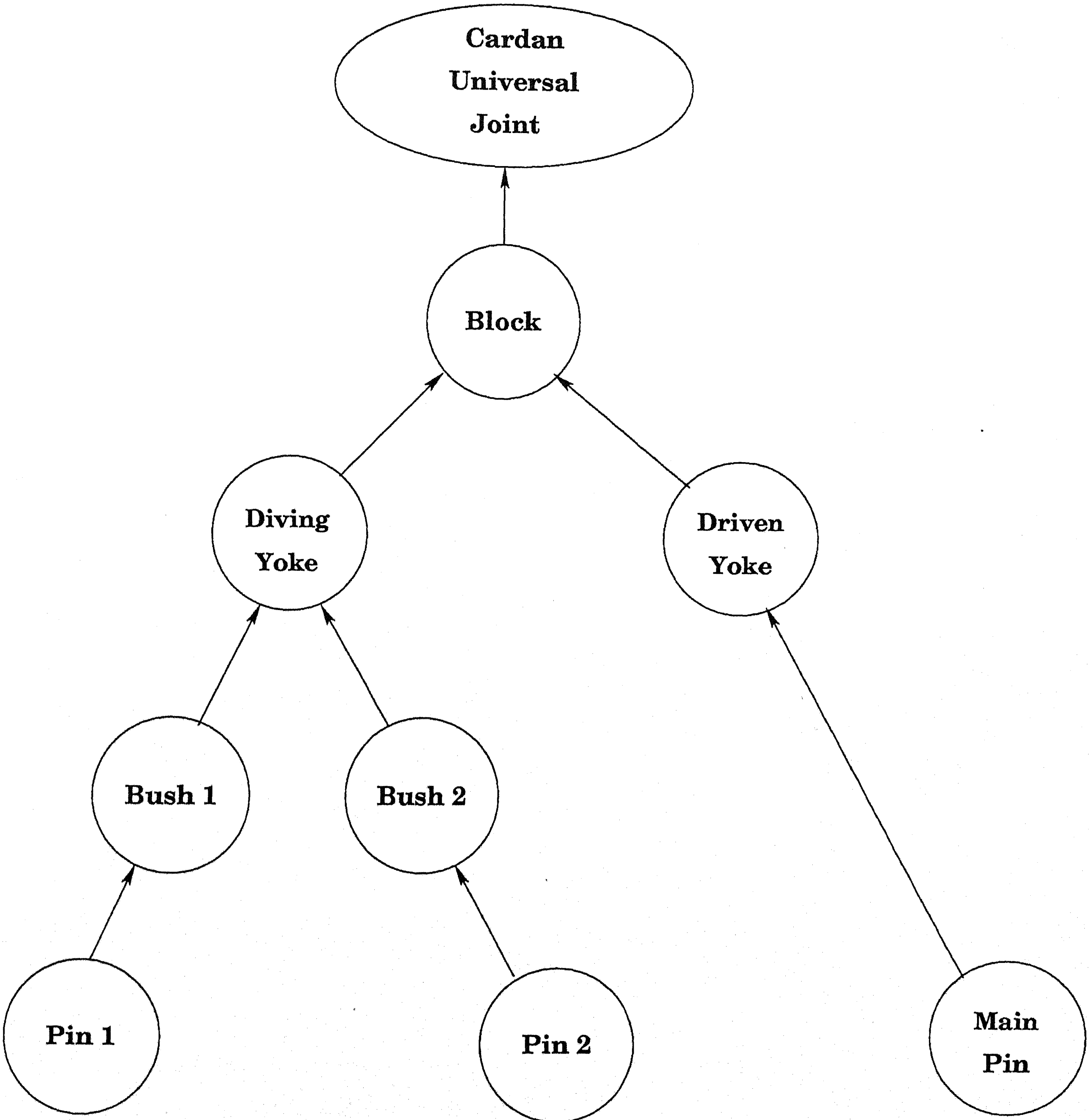

Figure 2.4: Assembly Tree for the Cardan Universal Joint

2.3.3 Specifying Mating Conditions

Individual Parts of an assembly are usually created together separately using a CAD/CAM system and then merged together to form the assembly. The parts may have to be scaled up or down before merging to fit properly into the assembly. Each part has its own database with its own MCS (Model Coordinate System). Typically, the user selects one of the parts as a base part and merges the other parts into it. The MCS of the host becomes the global coordinate system for this part.

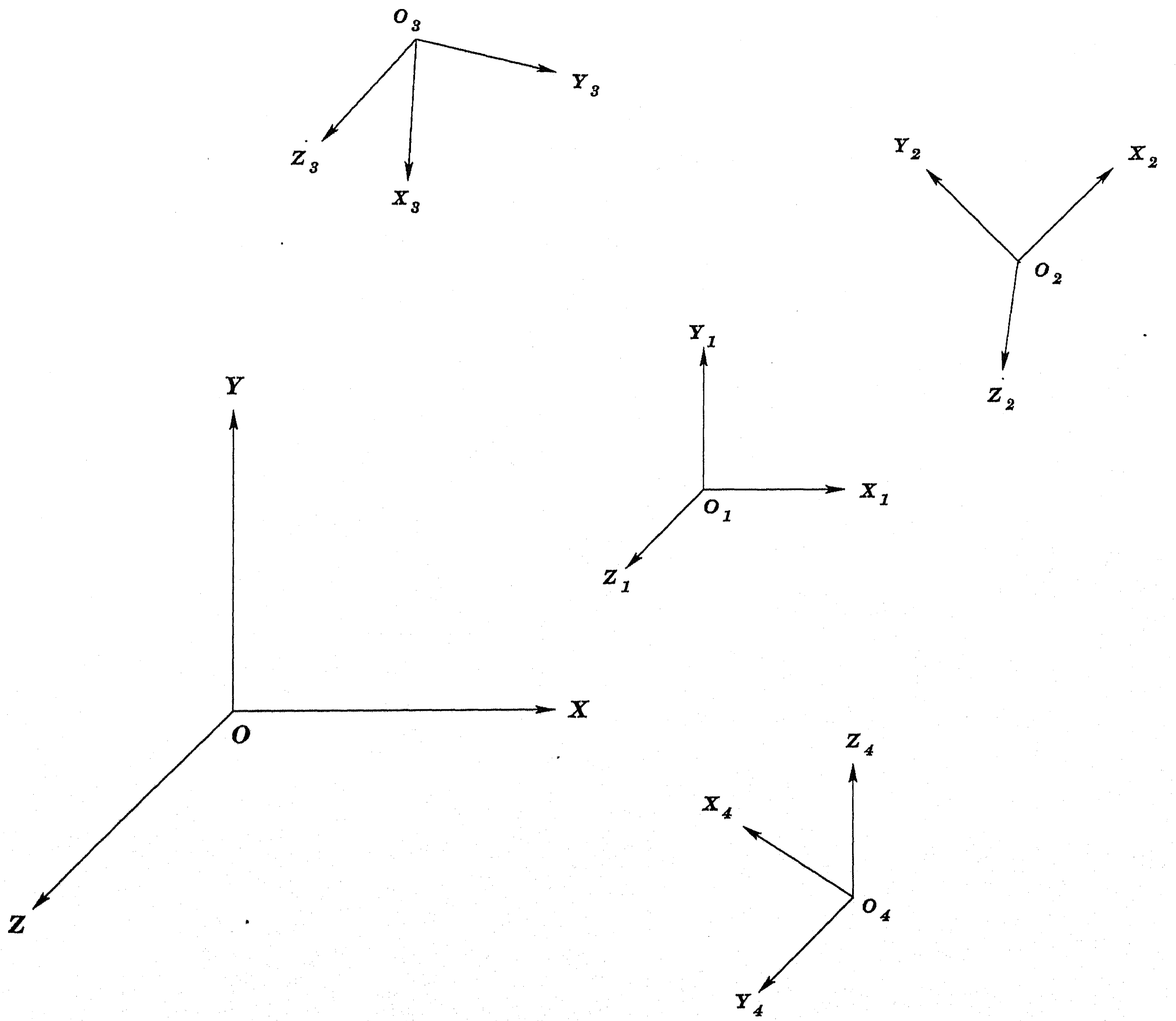

Figure 2.5: Positioning individual parts into their corresponding assembly

The final correct position of each part in the assembly is obtained by locating and orienting its corresponding MCS properly with respect to the global coordinate system of the assembly. Fig.(2.5) shows an example of positioning individual parts into their corresponding assembly. The XYZ is the global coordinate system of the assembly model with origin O.

The $X_1, Y_1, Z_1, X_2, Y_2, Z_2, X_3, Y_3, Z_3$ and X_4, Y_4, Z_4 are local coordinate systems of four parts that make the assembly. Their origins O_1, O_2, O_3 and O_4 are located properly relative to the assembly origin O and their orientations relative to the XYZ coordinate system reflect the proper orientation of the parts in their assembly.

2.4 Assembly Sequence Planning

As mentioned earlier, assembly may be defined as the process of fitting manufactured parts together into a complete machine, i.e., parts are joined together to form a functional unit. The fixing order of parts which is indirectly necessary to fit parts together to form a functional unit, is known as assembly sequence planning. Every functional component has one or more feasible assembly sequences. The development of feasible assembly sequence is a pre-task for product assembly planning. Few techniques exist to generate all assembly sequences, some of these techniques are manual while other are algorithmic.

Few references for assembly sequence planning are given in the Reference section

Chapter 3

Assembly Procedures

3.1 Approaches to represent an Assembly

There are two alternatives for representing an assembly depending on how the locations and orientations of its various parts are provided by the user. Accordingly, there are two approaches that exists to form assembly from a group of components.

1. Homogeneous Transformation Matrix Approach.

2. Mating Conditions Approach.

These two approaches are discussed in details in the following sections.

3.1.1 Homogeneous Transformation Matrix Approach

The simplest alternative to represent an assembly is to specify the location and orientation of each part in the assembly, together with the representation of the parts itself, by providing a 4×4 homogeneous transformation matrix. This matrix transforms the coordinates of the geometric entities of the part from its local coordinate system to the global coordinate system of the assembly. One convenient way for the user to provide the transformation matrix interactively is by specifying the location of the local coordinate system of a part relative to the assembly global coordinate system and by forcing the orientation of the local system to coincide with the orientation of the proper WCS (Working Coordinate System). This WCS is defined relative to the assembly global coordinate system.

As a WCS is completely defined by specifying its X and Y axes or its XY plane, the proper WCS used to merge a part into its assembly can be defined such that its XY plane coincides with the XY plane of the part MCS. In other words, the WCS becomes the part local coordinate system after merging and in effect the transformation matrix relates the part MCS to the host MCS (global coordinate system of the assembly). This alternative of merging parts into their assemblies is commonly used in various CAD/CAM systems.

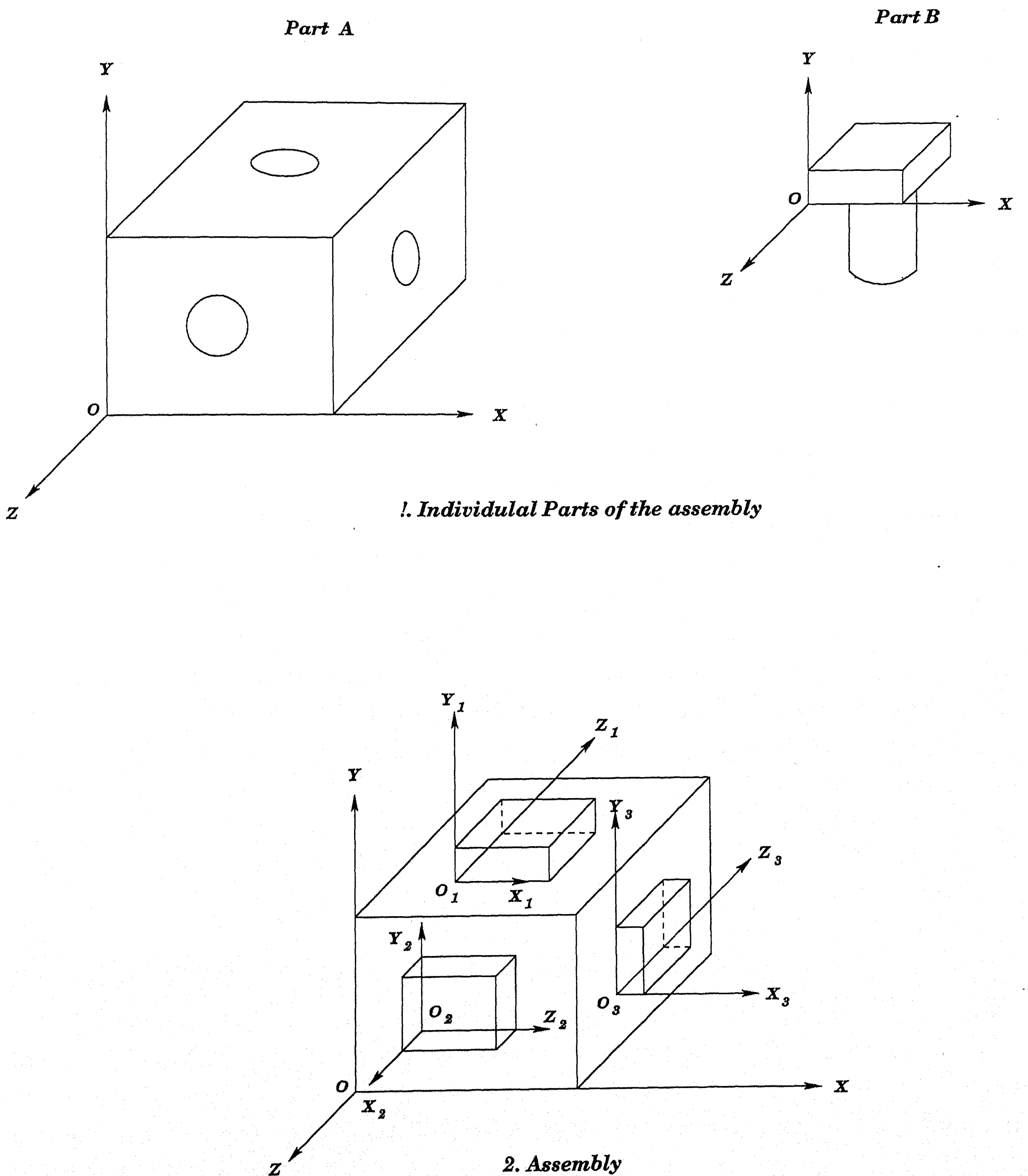

Figure 3.1: Problem of Instances

Let us consider an example of such instances as shown in Fig.(3.1). The assembly consists of two parts, A and B. Three instances of part B are used in the assembly. The user first provides the required information of both the parts with the MCS of each part as shown in the Fig.(3.1). Let us take part A as the base part. To create the assembly, the features of the host (holes on the faces) and the orientation of the instances must be properly aligned and then merge three instances of part B into it.

3.1.2 Mating Conditions Approach

This alternative to represent an assembly is based on specifying the spatial relationships between its individual parts as mating conditions. Mating feature information can be provided interactively with ease because mating features are simple graphic entities such as faces and center lines. For example, a mating condition can consists of planar faces butting up against each other. This is called as "against" condition. It may require the center lines of individual parts to be collinear. This is called as "fits" condition. Therefore, by simply identifying mating conditions the assembly data can be provided interactively. Providing the assembly data as mating features seems more natural than defining WCS required by the earlier approach.

By assigning the mating conditions, the transformation matrices that merge parts into their assembly can be automatically computed and stored for each part. In addition, using mating conditions instead of providing transformation matrices, determines whether or not the parts in assembly can be assembled. If some parts can not be physically assembled due to specifying inconsistent mating conditions, the transformation matrices satisfying all mating conditions do not exist. In this situation, the computation algorithm will diverge.

In most assemblies, the mating features between a pair of parts satisfy the conditions of "against," "fits," "tight fits," "contact," and "coplanar." A mating condition can be represented by its type and the two faces that mate. Soon we will discuss the relationship between the mating conditions and the actual position of the various parts.

Against condition:

The "against" condition holds between two planar faces, or between a planar face and a cylindrical face like shaft. This condition is illustrated in Fig.(3.2). Part 1 and part 2 have the MCSs as $X_1 Y_1 Z_1$ and $X_2 Y_2 Z_2$ respectively. The faces shown with the normal drawn are the faces to be mated. Each face is specified by its unit normal vector and any one point on the face with respect to the part MCS.
For example, the planar face of part 1 is specified by the unit normal $\hat{n}_1$, whose components are n_{1x}, n_{1y}, and n_{1z}, and by the point $P_1(x_1, y_1, z_1)$ with respect to the $X_1 Y_1 Z_1$ coordinate system. The "against" condition is satisfied by forcing $\hat{n}_1$ and $\hat{n}_2$ to be opposite to each other, and two faces to touch each other.

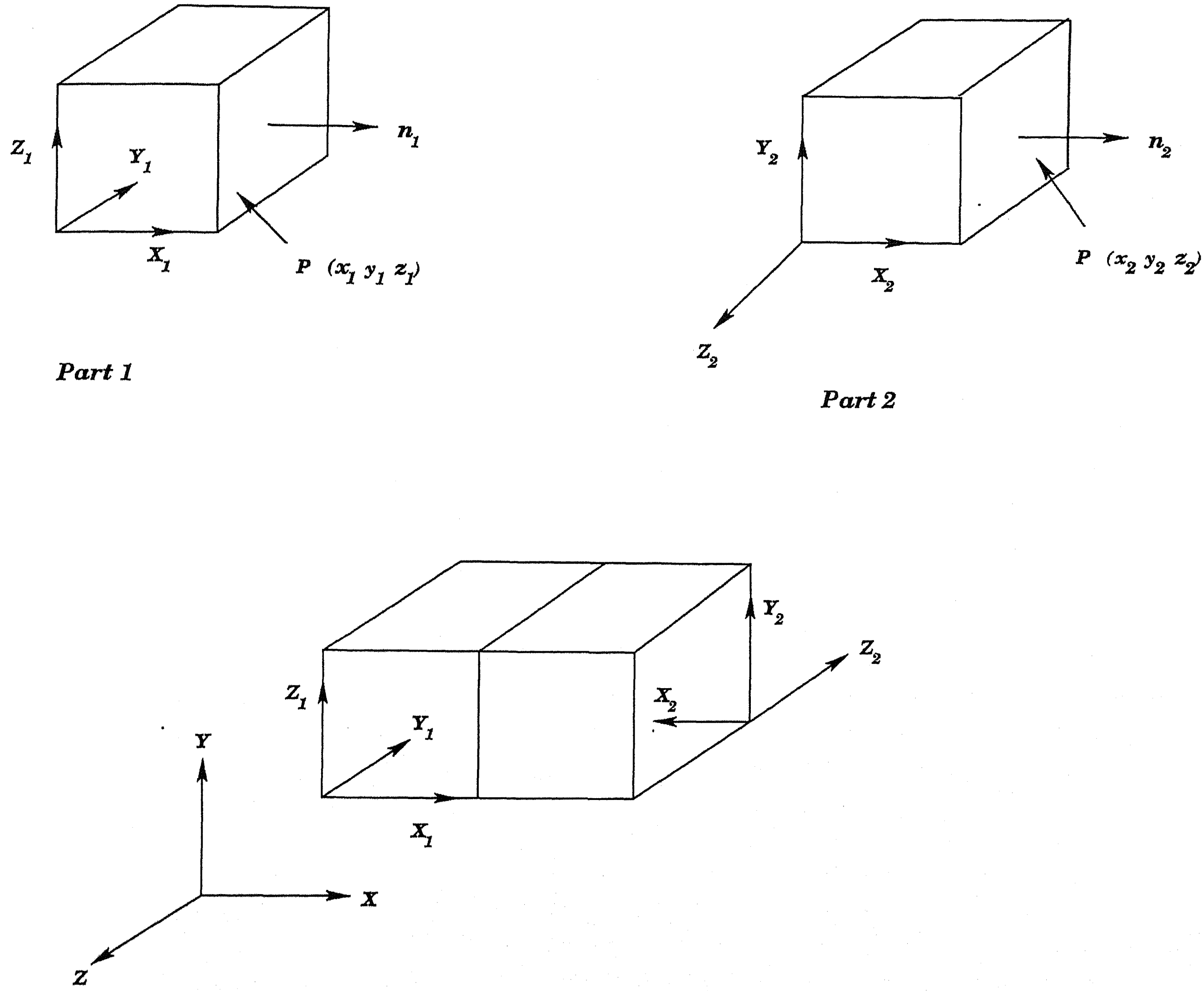

Figure 3.2: "Against" condition between two planar faces

Fits condition:

The "fits" condition holds between two cylindrical faces: a shaft cylindrical face and the hole cylindrical face as shown in Fig.(3.3). The "fits" condition is achieved by forcing the shaft and hole axes to be collinear. Each axis is specified by two points. The hole axis is specified by the two points $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ defined with respect to the $X_1\,Y_1\,Z_1$ coordinate system. Similarly, the shaft axis is specified by the two points $P_3(x_3, y_3, z_3)$ and $P_4(x_4, y_4, z_4)$ defined with respect to the $X_2\,Y_2\,Z_2$ coordinate system.

Contact and Tight fits condition:

The "against" and "fits" conditions as described above allow both translational and rotational freedom of movement between the mating parts. In the "against" condition shown in Fig.(3.2), part 2 can slide on part 1 or rotate relative to it after the two faces designated by the two normals $\hat{n}_1$ and $\hat{n}_2$ are mated together.

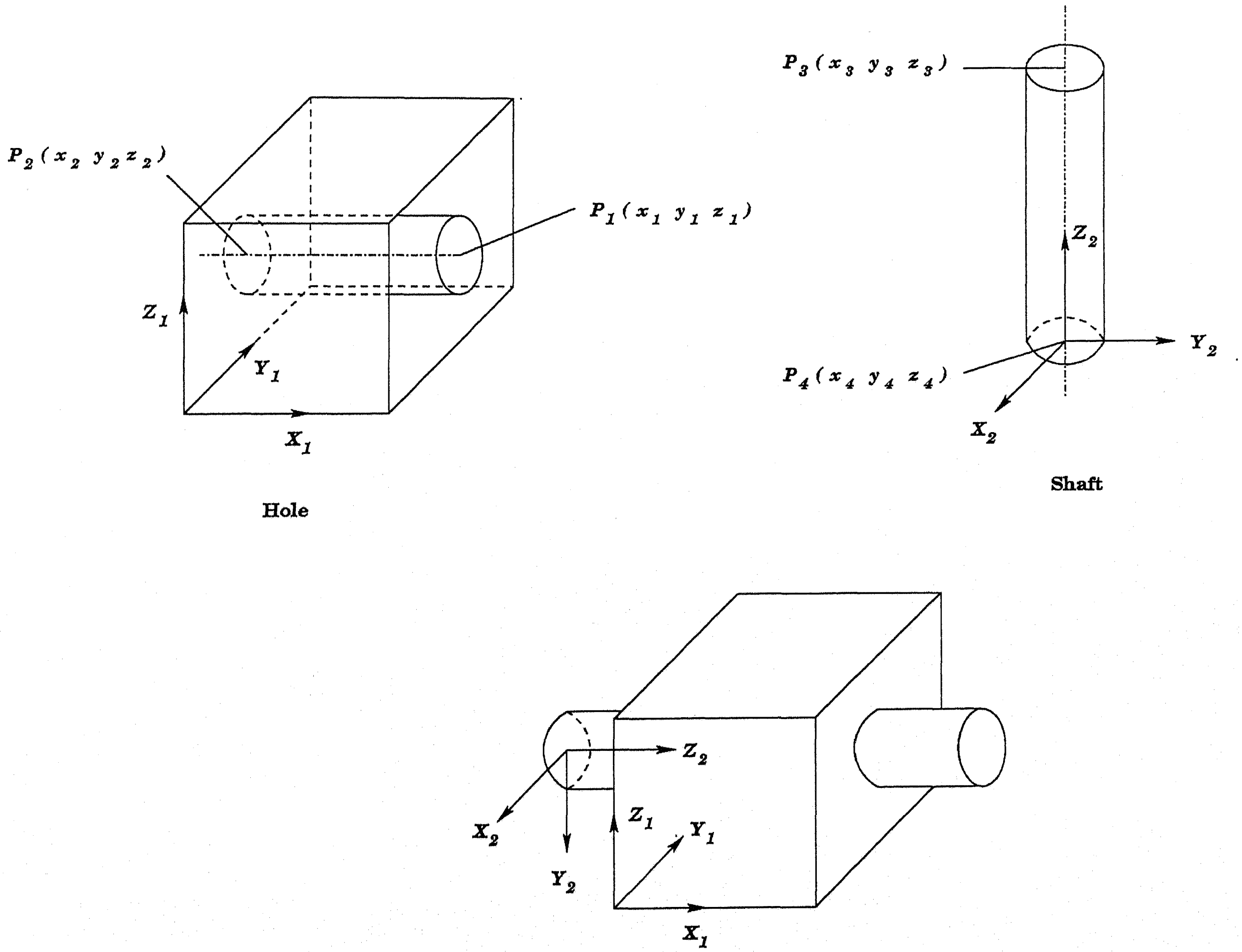

Figure 3.3: "Fits" condition

Similarly, in the "fits" condition shown in Fig.(3.3), the shaft can slide and/or rotate inside the hole. There are some parts with the "against" or "fits" condition where rotational, translational or both are not permitted. The "contact" and "tight fits" conditions are introduced to handle these cases. The "contact" condition prevents the freedom of movement due to the "against" condition and the "tight fits" prevents the freedom of movement due to the "fits" condition.

The "contact" condition is specified by requiring two points on the two mating parts to coincide. The "contact" condition does not exclude the "against" condition as former can allow rotation about the contact point. Consider the example shown in Fig.(3.4).

In this figure, faces are indicated indicated by the letter F with two subscripts. The first is the face number and second is the part to which the face belongs. For example, $F_{2,1}$ and $F_{1,2}$ are the second face of part 1 and the first face of part 2 respectively. Other points follow a similar convention as shown.

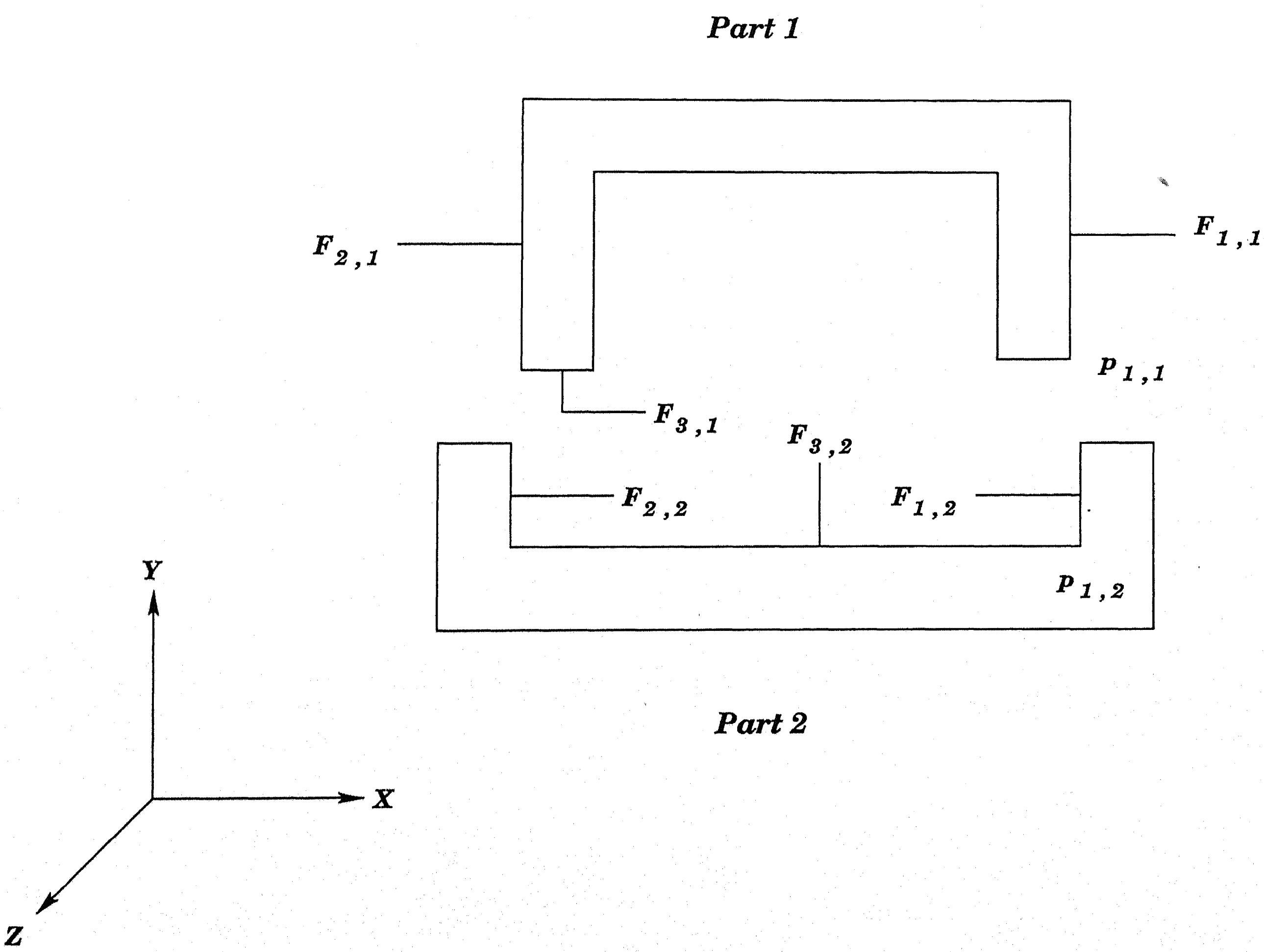

Figure 3.4: "Contact" condition

The mating conditions between these two parts can be specified by three "against" conditions: between $F_{1,1}$ and $F_{1,2}$, between $F_{2,1}$ and $F_{2,2}$ and between $F_{3,1}$ and $F_{3,2}$. With

these conditions specified only, part 1 should be free to move along the Z direction against $F_{3,2}$. If we specify the "contact" condition the points $P_{1,1}$ and $P_{1,2}$ are coincident, this undesired movement is eliminated. If only the "contact" condition is specified, part 1 can be tilted relative to part 2 and the proper faces may not be mated together.

The "tight fits" condition is introduced to prevent the rotational movements that may accompany the "fits" condition. The "tight fits" condition describes fits between parts where the force to rotate one part relative to another is too great to be called the rotational degree of freedom.

Coplanar condition:

The "coplanar" condition holds between two planar faces when they lie in the same plane. This condition is illustrated in Fig.(3.5). It is similar to the "against" condition except that the points P_1 and P_2 are chosen to lie on the two edges to mate. The "coplanar" condition is the complement of the "against" condition, and is satisfied by forcing the two normals $\hat{n}_1$ and $\hat{n}_2$ to be in the same direction.

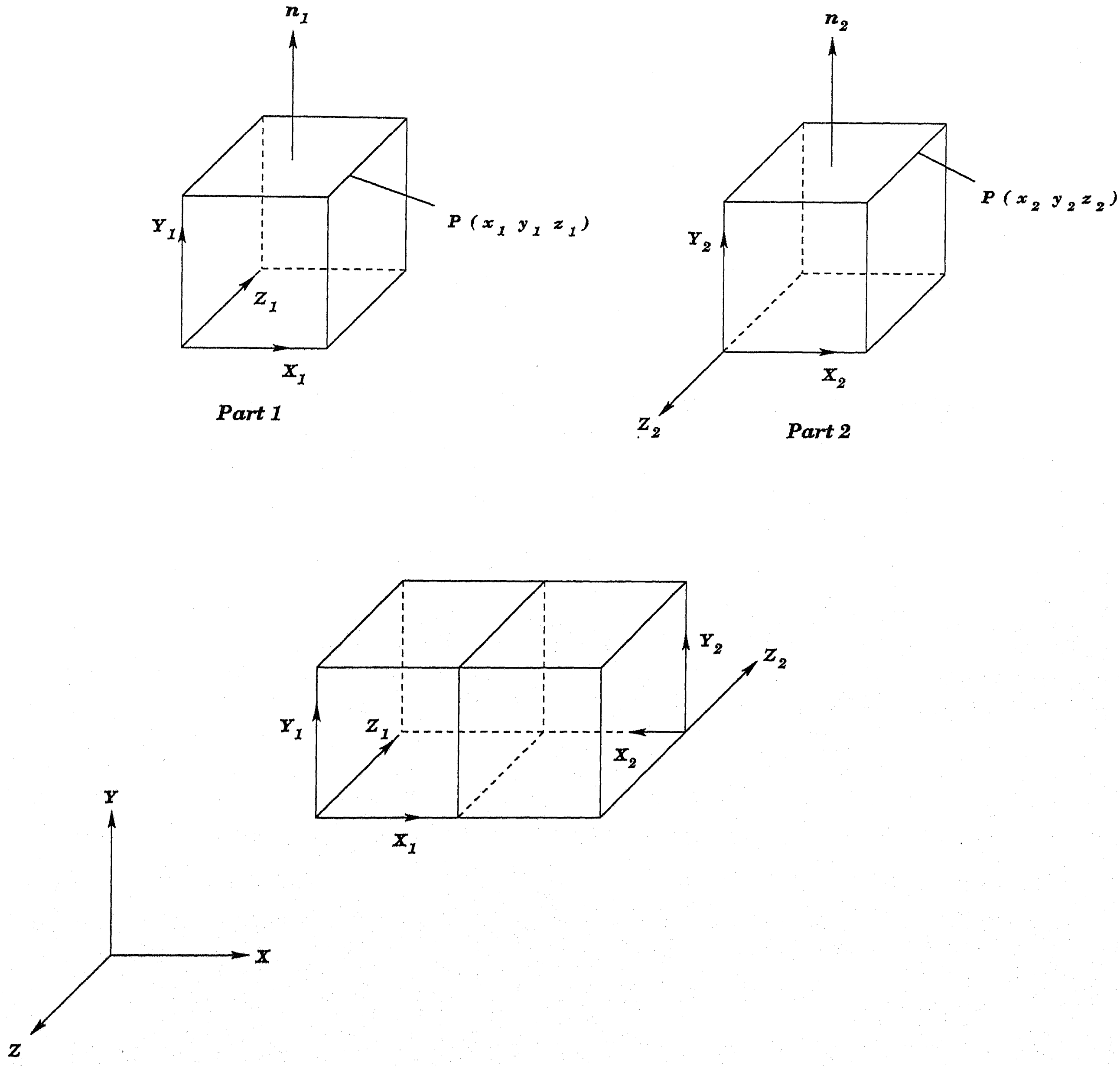

Figure 3.5: "Coplanar" condition

Chapter 4

Inferring Component Positions

Up till now we have seen that there are two approaches that can be used to represent an assembly. Now let us see, how we can infer the positions of various individual components in the assembly by the two approaches we discussed earlier.

4.1 Inferring positions by homogeneous transformation matrix approach

Before getting into the details of this approach, let us first understand what is homogeneous representation.

4.1.1 Homogeneous Representation

The various rigid-body geometric transformations are translation, scaling, mirroring and rotation. While transformations such as scaling, mirroring and rotation are in the form of matrix multiplication, translation takes the form of vector addition. This makes it inconvenient to concatenate the transformations involving translation. It is desirable, therefore, to express all the geometric transformations in the form of matrix multiplication only. Representing points by their homogeneous coordinates provides an effective way to unify the description of geometric transformations as matrix multiplications.

In homogeneous representation, an n-dimensional space is mapped into $(n+1)$-dimensional space, that is, a point (or a position vector) in n-dimensional space is represented by $(n+1)$ coordinates. In three-dimensional space, a point P with Cartesian coordinates (x, y, z) has the homogeneous coordinates (x^*, y^*, z^*, h), where h is any scalar factor $\neq 0$. The two types of coordinates are related to each other by the following equations:

$$x = \frac{x^*}{h} \qquad y = \frac{y^*}{h} \qquad z = \frac{z^*}{h} \tag{4.1}$$

Eqs.(4.1) are based on the fact that if the Cartesian coordinates of a given point P are multiplied by a scalar factor h, P is scaled to new point P^* and the coordinates of

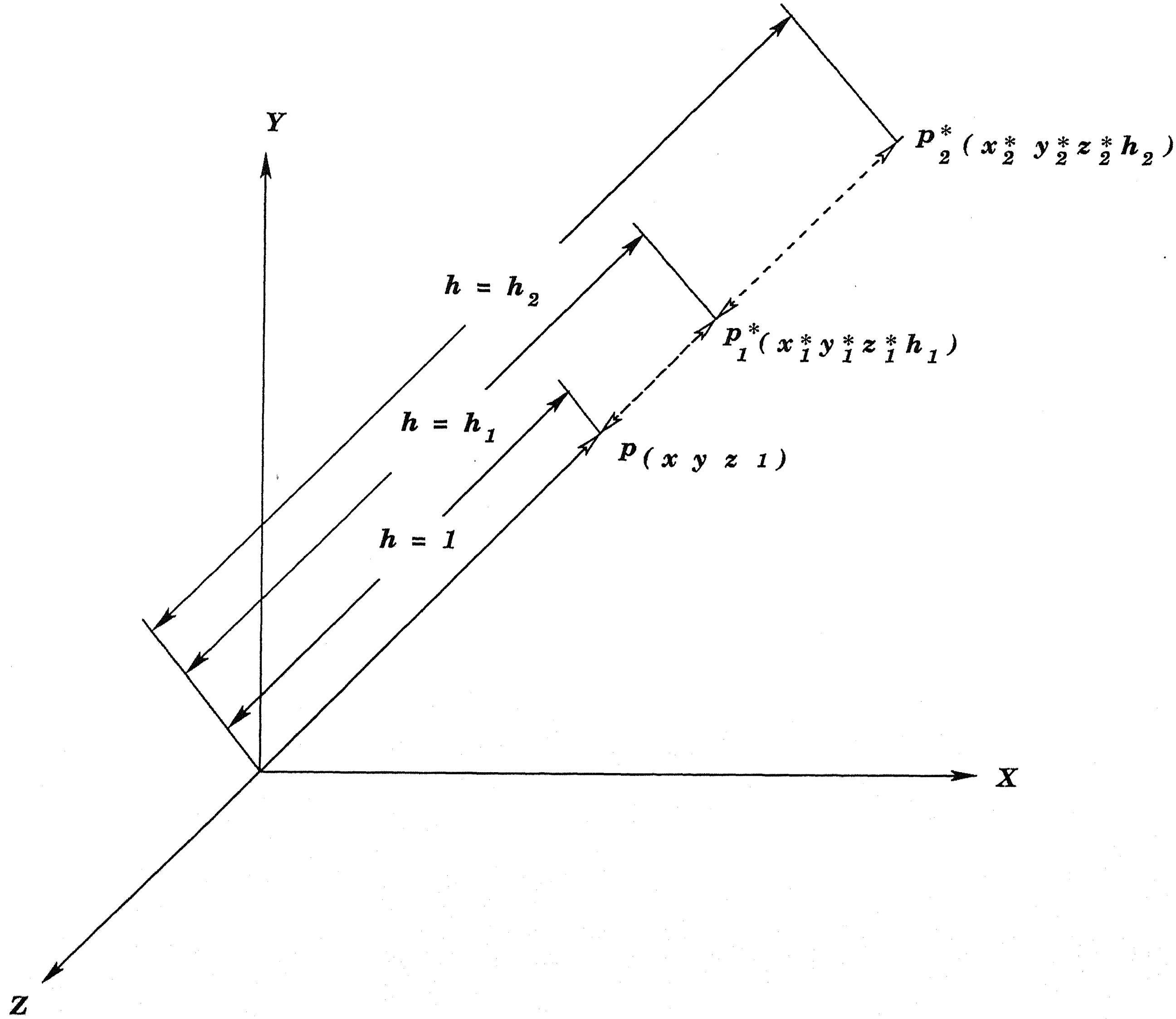

Figure 4.1: Homogeneous coordinates of point P

P and P^* are related by the above equations. Fig.(4.1) shows point P scaled by the two factors h_1 and h_2 to produce the two new points P_1^* and P_2^* respectively. These two points could be interpreted in two different ways. Once the Cartesian coordinates of P_1^* and P_2^* are calculated, their relationships to P do not exist any more. Moreover, the three points still belong to the Cartesian space. From a homogeneous coordinates point of view, the original point P is represented by $(x_1^*, y_1^*, z_1^*, h_1)$ and $(x_2^*, y_2^*, z_2^*, h_2)$ respectively according to Eqs.(4.1). More importantly, the three points belong to the homogeneous space, with the Cartesian coordinates obtained when $h = 1$, and the relationship between P and P_1^* or P_2^* is maintained through the proper value of h. For the purpose of geometric transformations, the scalar factor h used in Eqs.(4.1) is taken to be unity to avoid unnecessary division. Since translation and rotation are the major geometric transformations used in this work, we will discuss about them in short and see how they are achieved through the use of homogeneous coordinate system.

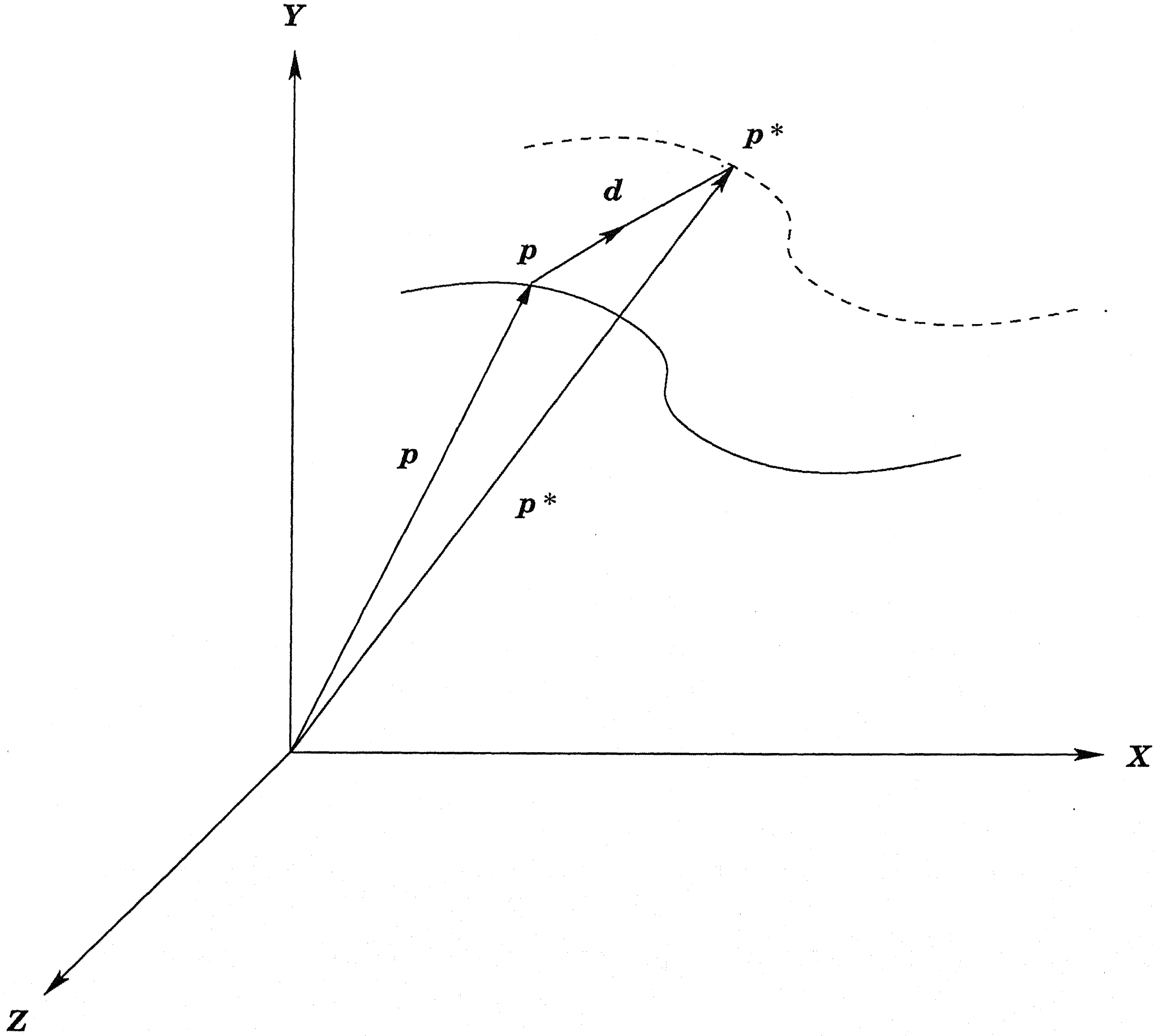

Figure 4.2: Translation of a curve

Translation:

When all the entities of a geometric model remains parallel to their original axes, the rigid body transformation is known as translation. Let us consider a point on a curve as shown in Fig.(4.2). This point is represented by a vector P. If we translate the curve by a vector d, the P occupies the new position as P^* as shown. This translation can be written in terms of vector addition as

$$P^* = P + d \tag{4.2}$$

As discussed earlier, it is always convenient to write the transformations in terms of matrices, the translation transformation given by Eq.(4.2) can now be written as matrix multiplication by adding the component of 1 to each vector in the equation and using a 4×4 matrix as

follows:

$$\begin{bmatrix} x^* & y^* & z^* & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_d \\ 0 & 1 & 0 & y_d \\ 0 & 0 & 1 & z_d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ y \\ 1 \end{bmatrix} \tag{4.3}$$

$$P^* = [D]P \tag{4.4}$$

Where $[D]$ is the translation matrix shown in Eq.(4.3).

Rotation:

Rotation is an important part of geometric transformation. It enables users to view geometric models from different angles and also helps many geometric operations. Rotation has unique characteristic that is not shared by translation, scaling, or reflection, that is, non-commutativeness. The final position and orientation of an entity after going through two subsequent translation, scaling, or reflection are independent of the operations, that is, commutative. On the contrary, two subsequent rotations of the entity about two different axes produce two different configurations depending on the order of rotations.

Rotation about coordinate system axes:

Rotating a point about a given angle θ about the X, Y, or Z axis is sometimes referred to as rotation about the origin.

To develop the rotational transformation of a point or a vector about one of the principal axes, let us consider the rotation of a point P at a positive angle θ about the Z axis, as shown in Fig.(4.3). This case is equivalent to two-dimensional rotation of a point in the XY plane about the origin. The final position of point P is shown as point P^*. The coordinates of the P^* can be written as

$$\begin{aligned} x^* &= r\cos(\theta+\alpha) = r\cos\alpha\cos\theta - r\sin\alpha\sin\theta \\ y^* &= r\sin(\theta+\alpha) = r\sin\alpha\cos\theta - r\cos\alpha\sin\theta \\ z^* &= z \end{aligned} \tag{4.5}$$

Where, $r = |P| = |P^*|$

To eliminate the angle θ from Eqs.(4.5), we can write (refer to the trigonometry in Fig.(4.3))

$$x = r\cos\alpha \qquad y = r\sin\alpha \tag{4.6}$$

Substituting Eqs.(4.6) into Eqs.(4.5) gives

$$x^* = x\cos\theta - x\sin\theta$$

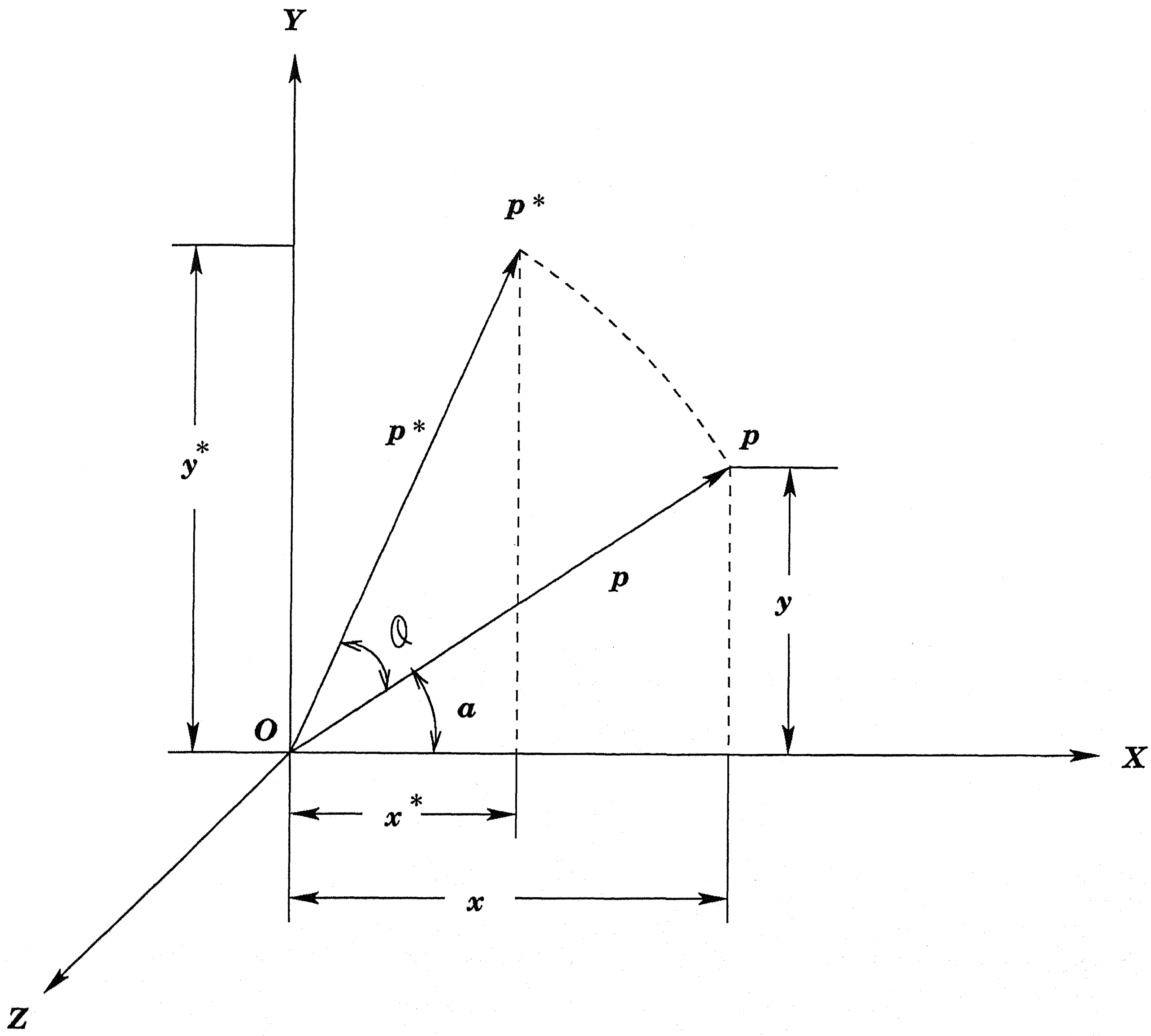

Figure 4.3: Rotation of a point about the Z axis

$$y^* = x\cos\theta - x\sin\theta \tag{4.7}$$

$$z^* = z$$

Rewriting Eqs.(4.7) in a matrix form gives

$$\begin{bmatrix} x^* \\ y^* \\ z^* \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ y \end{bmatrix} \tag{4.8}$$

or

$$P^* = [R_z]P \tag{4.9}$$

Where,

$$[R_z] = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

Similarly, we can prove that matrices for rotations about X and Y axes are given by

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{4.11}$$

$$[R_y] = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{4.12}$$

Thus in general, we can write

$$P^* = [R]P \tag{4.13}$$

Where [R] is the appropriate rotational matrix.

A closer look at the transformation matrices shows that they can all be embedded into one 4×4 matrix. This matrix takes the form:

$$[T] = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} = \begin{bmatrix} T_1 & T_2 \\ T_3 & 1 \end{bmatrix} \tag{4.14}$$

The 3×3 submatrix $[T_1]$ produces scaling, reflection, or rotation. the 3×1 column matrix $[T_2]$ generates translation. The 1×3 matrix $[T_3]$ produces perspective projection. The fourth diagonal element is the homogeneous-coordinates scalar factor h used in Eq.(4.1).

Concatenated Rotation:

Eqs.(4.10) to (4.12) shows the matrices for rotation about X, Y and Z axes respectively. Since series of rotations may be applied to a geometric model, combining or concatenating rotations is quite useful. Concatenated transformations are simply obtained by multiplying the $[T]$ matrices in Eq.(4.14). In order to derive the rotation submatrix in the homogeneous transformation matrix, let us consider a point, or its position vector, in the fixed coordinate system XYZ, that is, MCS, by the following rotations in the following order: α about Z axis, β about Y axis, and γ about the X axis. Substituting α, β and γ in Eqs.(4.10), (4.12) and (4.11) respectively and multiplying, we obtain the concatenated rotation matrix as

$$[T] = [T_x][T_y][T_z] \tag{4.15}$$

or

$$\begin{bmatrix} [R] & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} [R_x] & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} [R_y] & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} [R_z] & 0 \\ 0 & 1 \end{bmatrix} \tag{4.16}$$

or

$$[R] = [R_x][R_y][R_z] \tag{4.17}$$

Expanding the above equation gives

$$[R] = \begin{bmatrix} c\alpha c\beta & -s\alpha c\beta & s\beta \\ s\alpha c\gamma + c\alpha s\beta s\gamma & c\alpha c\gamma - s\alpha s\beta s\gamma & -c\beta s\gamma \\ s\alpha s\gamma - c\alpha s\beta c\gamma & c\alpha s\gamma + s\alpha s\beta c\gamma & c\beta c\gamma \end{bmatrix} \tag{4.18}$$

We are using Eq.(4.18) as $[T_1]$ in Eq.(4.14).

4.2 Inferring positions from mating conditions

Figs.(3.2), (3.3), (3.5) show that satisfying mating conditions requires repositioning of the parts of the assembly. The inference of the location and orientation of a part in an assembly from mating conditions requires computing its transformation matrix from these conditions. This matrix relates the part's local coordinate system to the global coordinate system of the assembly. With reference to Fig.(2.5), the location of part 1 is represented by the vector OO_1 connecting the origin O of the assembly global coordinate system to the origin O_1 of the $X_1 Y_1 Z_1$ local coordinate system of the part. The orientation is represented by the rotation matrix between the two systems. The transformation matrix can be written as

$$[T] = \begin{bmatrix} m_x & q_x & r_x & x \\ m_y & q_y & r_y & y \\ m_z & q_z & r_z & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

This matrix has twelve variables. Out of twelve variables, nine are rotational and three are translational variables. These variables must be determined from mating conditions. For an assembly of N parts, and choosing one of them as base part, $N-1$ transformation matrices have to be computed. Therefore, variables to solve for simultaneously are the $12 \times (N-1)$ elements of these matrices.

In a typical assembly, the mating conditions between two components are not enough by themselves to completely constrain the two components. An intertwinement of mating conditions usually exists between all of the parts. In general, a group of parts must be solved simultaneously. The mating conditions along with the transformation matrix properties provide the constraint equations necessary to solve for the $12 \times (N-1)$ variables. The number of equations is always equal to or greater than the number of variables. Therefore the method of solution should account for the number of redundant equations, and eliminate these equation from the system of equations to be solved.

Before discussing the algorithms to solve for the unknown variables, the development of constraint equations from mating conditions is presented. We will discuss the three basic the three basic mating conditions: "against," "fits," and "coplanar." For the "against" condition

shown in Fig.(3.2), each face where the two part mate is specified by a unit normal and a point described in the local coordinate system of its corresponding part. Let $[T_1]$ and $[T_2]$ be the transformation matrices from the $X_1Y_1Z_1$ and $X_2Y_2Z_2$ coordinate system respectively to the global coordinate system of the assembly. The unit normals and two points specifying the mating conditions can be expressed in terms of the XYZ system as follows:

$$\begin{bmatrix} n_{1x}^a \\ n_{1y}^a \\ n_{1z}^a \\ 0 \end{bmatrix} = [T_1] \begin{bmatrix} n_{1x} \\ n_{1y} \\ n_{1z} \\ 0 \end{bmatrix} \tag{4.20}$$

$$\begin{bmatrix} {x_1}^a \\ {y_1}^a \\ {z_1}^a \\ 1 \end{bmatrix} = [T_1] \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 0 \end{bmatrix} \tag{4.21}$$

$$\begin{bmatrix} n_{2x}^a \\ n_{2y}^a \\ n_{2z}^a \\ 0 \end{bmatrix} = [T_2] \begin{bmatrix} n_{2x} \\ n_{2y} \\ n_{2z} \\ 0 \end{bmatrix} \tag{4.22}$$

and

$$\begin{bmatrix} {x_2}^a \\ {y_2}^a \\ {z_2}^a \\ 1 \end{bmatrix} = [T_2] \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 0 \end{bmatrix} \tag{4.23}$$

In the above equations, the superscript a indicates assembly. The "against" condition requires the direction of the two unit normals to be equal and opposite and the two points to lie in the same plane at which the two faces mate. These can be expressed by the following four equations:

$$n_{1x}^a = -n_{2x}^a \tag{4.24}$$

$$n_{1y}^a = -n_{2y}^a \tag{4.25}$$

$$n_{1z}^a = -n_{2z}^a \tag{4.26}$$

$$\begin{bmatrix} n_{1x}^a \; n_{1y}^a \; n_{1z}^a \; 0 \end{bmatrix} \left\{ \begin{bmatrix} x_1^a \\ y_1^a \\ z_1^a \\ 0 \end{bmatrix} - \begin{bmatrix} x_2^a \\ y_2^a \\ z_2^a \\ 0 \end{bmatrix} \right\} = 0 \tag{4.27}$$

Hence, four equations [(4.24) to (4.27)] are required for each "against" condition to be satisfied.

The "fits" condition requires that the center lines of the shaft and the hole be collinear as shown in Fig.(3.3). The equation of the centerline can be written as

$$\frac{x - x_1^a}{x_2^a - x_1^a} = \frac{y - y_1^a}{y_2^a - y_1^a} = \frac{x - z_1^a}{z_2^a - z_1^a} \tag{4.28}$$

If the shaft axis is collinear with the hole centerline, points P_3 and P_4 defining the axis should satisfy Eq.(4.28). The points must first be transformed using $[T_2]$ to the assembly global coordinate system. The constraint equations required for each "fits" condition can be written as

$$\frac{x_3^a - x_1^a}{x_2^a - x_1^a} = \frac{y_3^a - y_1^a}{y_2^a - y_1^a} = \frac{x_3^a - z_1^a}{z_2^a - z_1^a} \tag{4.29}$$

and

$$\frac{x_4^a - x_1^a}{x_2^a - x_1^a} = \frac{y_4^a - y_1^a}{y_2^a - y_1^a} = \frac{x_4^a - z_1^a}{z_2^a - z_1^a} \tag{4.30}$$

Each of the above equations yields three combinations of equations resulting in a total of six equations for each "fits" condition. In general, two of these equations are redundant because Eqs.(4.29) and (4.30) each yields only two independent equations instead of three. however it is necessary to carry out all the three to cover the cases where the centerline passing through points P_1 and P_2 is parallel to any of the global coordinate axes. For example, if the centerline is parallel to the X axis as shown in Fig.(3.3), Eq.(4.29) becomes

$$\frac{x_3^a - x_1^a}{x_2^a - x_1^a} = \frac{y_3^a - y_1^a}{0} = \frac{x_3^a - z_1^a}{0} \tag{4.31}$$

Which gives the following two equations only:

$$(y_3^a - y_1^a)(x_2^a - x_1^a) = 0 \tag{4.32}$$

$$(z_3^a - z_1^a)(x_2^a - x_1^a) = 0 \tag{4.33}$$

Hence, it can be seen that all three equations must be carried out so that at least two independent equations can be written for all cases, although this introduces redundancy in the system of equations.

The constraint equations for the "coplanar" condition are the same as for the "against" condition except that the two unit normals have to be in the same direction as shown in Fig.(3.5). Thus Eqs.(4.24) to (4.27) can be used after replacing the minus sign in Eqs.(4.24) to (4.26) with a plus sign.

One last constraint equation can be written and applies to all free rotating parts in the assembly such as bolts, pins, shafts, etc. A free rotating part is defined here as a part that rotates freely about a centerline axis. The rotation of these parts usually does not alter the appearance of the assembly. Thus this condition is called as "free rotation." Therefore,

there can be infinitely possible orientations of a free rotating part. Fig.(4.1) shows a bolt with its $X_1Y_1Z_1$ local coordinate system oriented in the XYZ global coordinate system of the assembly. The bolt can rotate freely about its X_1 axis by an angle ϕ. As seen from the figure, the orientation of the Y_1 and Z_1 axes are insignificant to the assembly. If this free rotation is not constrained, the calculations of the transformation matrix from the mating conditions will diverge. As long as the angle ϕ is arbitrary, it can be set to zero, that is,

$$\phi = 0 \tag{4.34}$$

This constraint equation represents the constraint equation associated with free rotating parts. With reference to Fig.(4.4), Eq.(4.34) is equivalent to two equations:

$$a = b = 0 \tag{4.35}$$

Where a and b are the components of the unit vectors (along the Y_1 and Z_1 axes) in the Z_1 and Y_1 directions respectively. If the X_1 axis is coincident with the X axis of the global coordinate system of the assembly, Eq.(4.35) can be written in terms of the elements of the transformation matrix given by Eq.(4.19) as

$$q_z = r_y = 0 \tag{4.36}$$

For a part rotating freely about its Y_1 axis or Z_1 axis, we can write respectively

$$m_z = r_x = 0 \tag{4.37}$$

$$m_y = q_x = 0 \tag{4.38}$$

With all the constraint equations derived for the various mating conditions, we can now calculate the total number of equations and unknowns that can be used to infer the positions of a part from mating conditions. For each "against" condition, 16 equation can be written: 12 are provided by Eqs.(4.20) to (4.23) and the other 4 Eqs. (4.24) to (4.27). For each "fits" condition, 18 equations can be written: 12 are provided by Eqs.(4.20) to (4.23) and the other 6 are Eqs.(4.29) to (4.30). The "coplanar" condition provides the same number of equations as the "against" condition. For each free rotating part, two equations [Eqs.(4.36), (4.37), or (4.38)] are available. In addition, the properties of the transformation matrix [Eq.(4.19)] provides six equations: three from the unit vector length property and three from the orthogonality property. These can be written as

$$m_x^2 + m_y^2 + m_z^2 = 1 \tag{4.39}$$

$$q_x^2 + q_y^2 + q_z^2 = 1 \tag{4.40}$$

$$m_x q_x + m_y q_y + m_z q_z = 0 \tag{4.41}$$

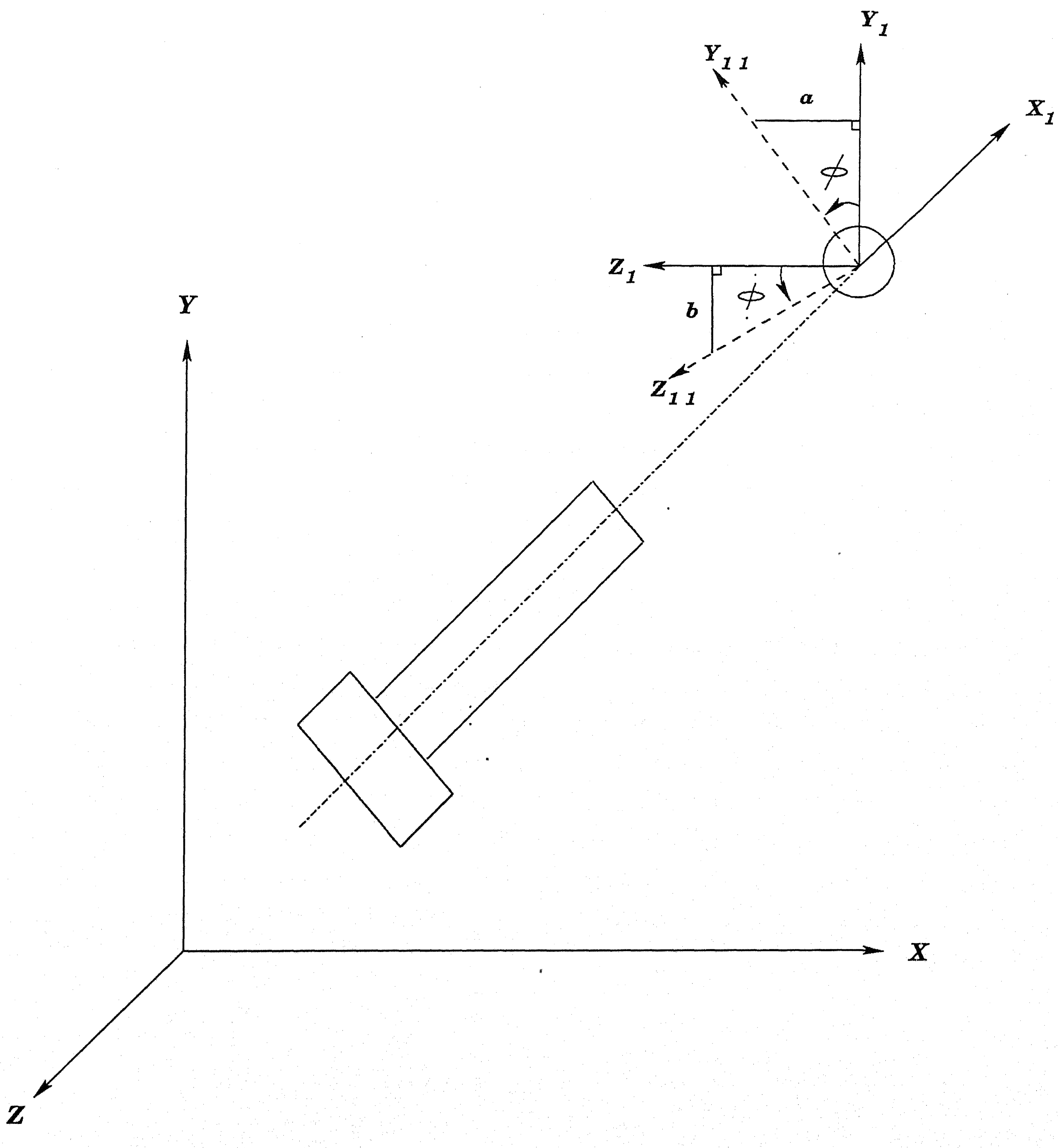

Figure 4.4: "Free rotating part" condition

$$r_x = m_y q_z - m_z q_y \tag{4.42}$$

$$r_y = m_z q_x - m_x q_z \tag{4.43}$$

$$r_z = m_x q_y - m_y q_x \tag{4.44}$$

For an assembly of N parts, the total number of equations that can be written is given by

$$M = 6(N-1) + 16NA + 16NC + 18NF + 2NR \tag{4.45}$$

Where NA, NC, NF and NR are respectively the number of "against," "coplanar," "fits," and "free rotation" conditions. The number of variables are

$$V = 12(N-1) + 12(NA + NC + NF) \tag{4.46}$$

The first term of this equation represents the elements of the $(N-1)$ transformation matrices and the second term is the number of variables introduced by the "against," "coplanar," "fits," conditions. Each condition introduces 12 variables: 6 components of the unit normal vectors $\hat{n}_1$ and $\hat{n}_2$ and 6 coordinates of the two points P_1 and P_2 (or P_3 and P_4) for the "fits" condition). Equations (4.45) and (4.46) show that the number of equations and the number of variables are not equal. In general, the former is always equal to or greater than the latter.

The representation of the transformation matrix by Eq.(4.19) increases both the number of variables and the number of equations, which in turn makes finding a solution an expensive proposition. In an effort to reduce the number of variables, Eq.(4.19) may be rewritten using the following approach. The position of a part is still given by the elements x, y and z. The orientation can be described by a sequence of rotation about the X, Y and Z axes instead of using the components of the unit vectors as given by the nine elements of the rotation matrix. Let us assume that the local coordinate system of a part can be oriented properly in its assembly by three rotations about the axes of the global coordinate system in the following order: α about the Z axis, β about the Y axis, and γ about the X axis. Thus, we can rewrite Eq.(4.19) as

$$[T] = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & [R] & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.47}$$

Where $[R]$ is given by Eq.(4.18).

Using Eq.(4.47) instead of Eq.(4.19) reduces the number of variables per matrix almost by half: from 12 to 6 and they are α, β, γ, x, y, z. Consequently, six equations[Eqs.(4.39) to (4.44)] are eliminated. In addition only one constraint equation [Eq.(4.34)] is used for free rotation. To reduce the number of equations and variables even further, we can eliminate the 12 variables (global description of unit normals and points) given by Eqs.(4.20) to (4.23) by

considering them known in terms of the transformation matrix. Thus, Eqs.(4.45) and (4.46) become respectively

$$M = 4NA + 4NC + 6NF + NR \tag{4.48}$$

$$V = 6(N-1) \tag{4.49}$$

To simplify the implementation of the free rotation condition, we can define the Z as the axis of free rotation. Therefore the angle ϕ in Eq.(4.34) is about this axis. If the free rotating axis is the Y axis, we will assume rotation about this axis which is given by γ to be zero.

Chapter 5

Solution Schemes

5.1 Homogeneous Transformation Matrix Approach

As mentioned earlier, the simplest alternative of representing an assembly is to specify the location and orientation of each part in the assembly, together with the representation of the part itself, by providing a 4×4 homogeneous transformation matrix. Homogeneous transformation matrix is given by Eq.(4.19). Where, location is obtained by three translational elements and orientation by 9 rotational elements. We have also seen that the 9 rotational elements can always be described by a sequence of rotations about the X, Y, or Z axes instead of using the components of the unit vectors as given by the nine elements of rotation matrix. Rewriting Eq.(4.14),

$$[T] = \begin{bmatrix} T_1 & T_2 \\ T_3 & 1 \end{bmatrix} \tag{5.1}$$

Where, $[T_1]$ is the rotational matrix given by Eq.(4.18), $[T_2]$ is a translational 3×1 column matrix, $[T_3]$ is 1×3 row matrix producing the perspective projection. Here, for convenience, we will assume all the elements of $[T_3]$ to be zero. After proper substitution, the homogeneous transformation matrix becomes

$$[T] = \begin{bmatrix} c\alpha c\beta & -s\alpha c\beta & s\beta & x \\ s\alpha c\gamma + c\alpha s\beta s\gamma & c\alpha c\gamma - s\alpha s\beta s\gamma & -c\beta s\gamma & y \\ s\alpha s\gamma - c\alpha s\beta c\gamma & c\alpha s\gamma + s\alpha s\beta c\gamma & c\beta c\gamma & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Now, let us assume that we have represented all the components of the assembly in a WCS in such a way that for any two parts, which are going to assemble, their faces are against each other, that is, their corresponding normals are opposite to each other. Under this situation orientation of the local coordinate system of the parts local coordinate system is same as the global coordinate system of the base part. So we need not compute α, β and γ variables in homogeneous transformation matrix given by Eq.(5.2). Therefore, we will assume these

variables to be zero. Then, Eq.(5.2) becomes

$$[T] = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

Let us consider a point P which after translation occupies a new position as P^*. Then, P and P^* are related by the following equation.

$$P^* = [T]P \tag{5.4}$$

For any general assembly problem, since we know the final position of the part in the assembly together with its initial position, we can say that we know P^* and P. So our task boils down to finding x, y and z component of the Eq.(5.3) using Eq.(5.4).
The solution scheme is given below

Let us consider S and I as a 4×1 column matrices given as follows

$$S = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad I = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

For convenience, let us substitute $x = y = z = 1$ in Eq.(5.3) temporarily, we get $[T]$ as

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since we know P and P^*, we will designate the product of $[T]$ and P as TP [from Eq.(5.4)] which is 4×1 column matrix. Then we get S, which is our solution vector by following equation

$$S = P^* - TP - I$$

Thus, we obtain the values of variables x, y and z.

In few assembly cases, it may so happen that the faces of the mating parts may not be opposite to each other although their corresponding coordinate systems are parallel to each other. In such situations, we will compare the normals of the corresponding faces of the two components of the assembly and rotate one of the component to make their faces opposite.

5.2 Solution Scheme for Mating Conditions Approach

We will now discuss the solution scheme for the system of equations that result from applying the mating conditions. This system is nonlinear due to the trigonometric functions that appear in the transformation matrix $[T]$. Since the number of equations is equal to or exceeds the number of variables, a method is needed to remove the redundant equations. The method discussed here utilizes the least square technique to eliminate the redundancy first, followed by using the Newton Raphson iteration method to solve the resulting set of independent equations. The Newton-Raphson method for n nonlinear equations of n variables can be written as

$$X_{k+1} = X_k + [J(X_k)]^{-1} R_k \tag{5.5}$$

Where X_k is the solution vector at the k^{th} iteration, $[J(X_k)]$ is the Jacobian matrix and R_k is the residual vector. Both of which are evaluated at the current solution vector X_k.

When redundancy exists, the inverse of the Jacobian may not exist because the Jacobian itself may not be square and/or it may be singular. The following procedure can be used to solve for n variables ($X = [x_1 \;\; x_2 \ldots x_n]^T$) using the following m equations:

$$\begin{gathered} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_m(x_1, x_2, \ldots, x_n) = 0 \end{gathered} \tag{5.6}$$

In vector form, Eqs.(5.6) becomes

$$F(X) = 0 \tag{5.7}$$

To write Eqs.(5.6) in Newton-Raphson iterative form, let us assume that a solution X_i exists at step i and the solution at step $i+1$ is X_{i+1} such that

$$X_{i+1} = X_i + \Delta X_i \tag{5.8}$$

Linearizing Eqs.(5.8) about X_i gives

$$F_{i+1} = F_i + \frac{\partial F(X_i)}{\partial X} \Delta X_i \tag{5.9}$$

If X_{i+1} is the solution, then Eq.(5.7) holds, that is, $F_{i+1} = 0$. Thus Eq.(5.9) becomes

$$\frac{\partial F(X_i)}{\partial X} \Delta X_i = -F_i \tag{5.10}$$

Expanding Eq.(5.10) gives:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_n} \\ & \vdots & & \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_i \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix}_i = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_m(x_1, x_2, \ldots, x_n) \end{bmatrix}_i \tag{5.11}$$

This can be written as:

$$[J]_i \Delta X_i = R_i \tag{5.12}$$

Where $[J]_i = [J(X_i)]$, ΔX_i, and R_i are the Jacobian matrix, the incremental solution vector and the residual vector at iteration i respectively. The Jacobian $[J(X_i)]$ is a nonsquare matrix of size $m \times n$.

The least-squares method may be used to solve Eq.(5.12) for ΔX_i. The method is based on multiplying both sides of Eq.(5.12) by $[J]_i{}^T$, that is,

$$[J]_i{}^T [J]_i \Delta X_i = [J]_i{}^T R_i \tag{5.13}$$

Solving this equation for ΔX_i gives

$$\Delta X_i = [J^T J]_i{}^{-1} [J]_i{}^T R_i \tag{5.14}$$

The algorithm to solve for ΔXi can described as follows:
An initial guess X_0 is made. The Jacobian $[J]_0$ and the residual vector R_0 are computed. Next Eq.(5.14) is used to calculate ΔX_0. Finally, Eq.(5.8) is used to compute X_1. These steps are repeated to obtain ΔX_1 and X_2, ΔX_2 and X_3,..., and ΔX_{n-1} and X_n. Convergence is achieved when the elements of the residual vector R or the incremental solution vector ΔX approaches zero.

- We have already mentioned that convergence is obtained when the residual or incremental vector approaches zero. In the present work, for all the cases solved, if we calculate R_1, we find it to be zero, therefore $\Delta X_1 = 0$ and $X_2 = X_1$, Thus X_1 is the solution. Actually, it is the exact solution. Therefore, we do not go for further iterations using Newton-Raphson method.

Chapter 6

Results and Discussions

6.1 Cases Solved

It is always useful when studying the assembly of a product to identify the various ways in which the assembly process may be carried out. In most assemblies, there are more than one assembling sequence to generate assemblies from their respective parts. An assembly or production engineer must decide on the most optimum sequence.

Few techniques exist to generate all assembly sequences, some of these techniques are manual while other are algorithmic. A *precedence diagram* may be designed to show all the possible assembly sequences. In this work, although there are many possible sequences, a most convenient assembly sequence has been followed.

Following cases have been solved successfully using both the approaches. In all the cases a manual assembly sequence is being used.

1. Cardan Universal Joint Assembly.
2. Clamp Assembly.
3. Protective Flange Coupling Assembly.
4. Problem involving different mating conditions.
5. Problem of instances.

In the first three cases, we have drawn the assembly tree that shows the spatial relationships between various components of assembly. For all the cases, no tolerances are provided on dimensions of the components and the fits between the various components of an assembly are assumed to be "interference" fits.

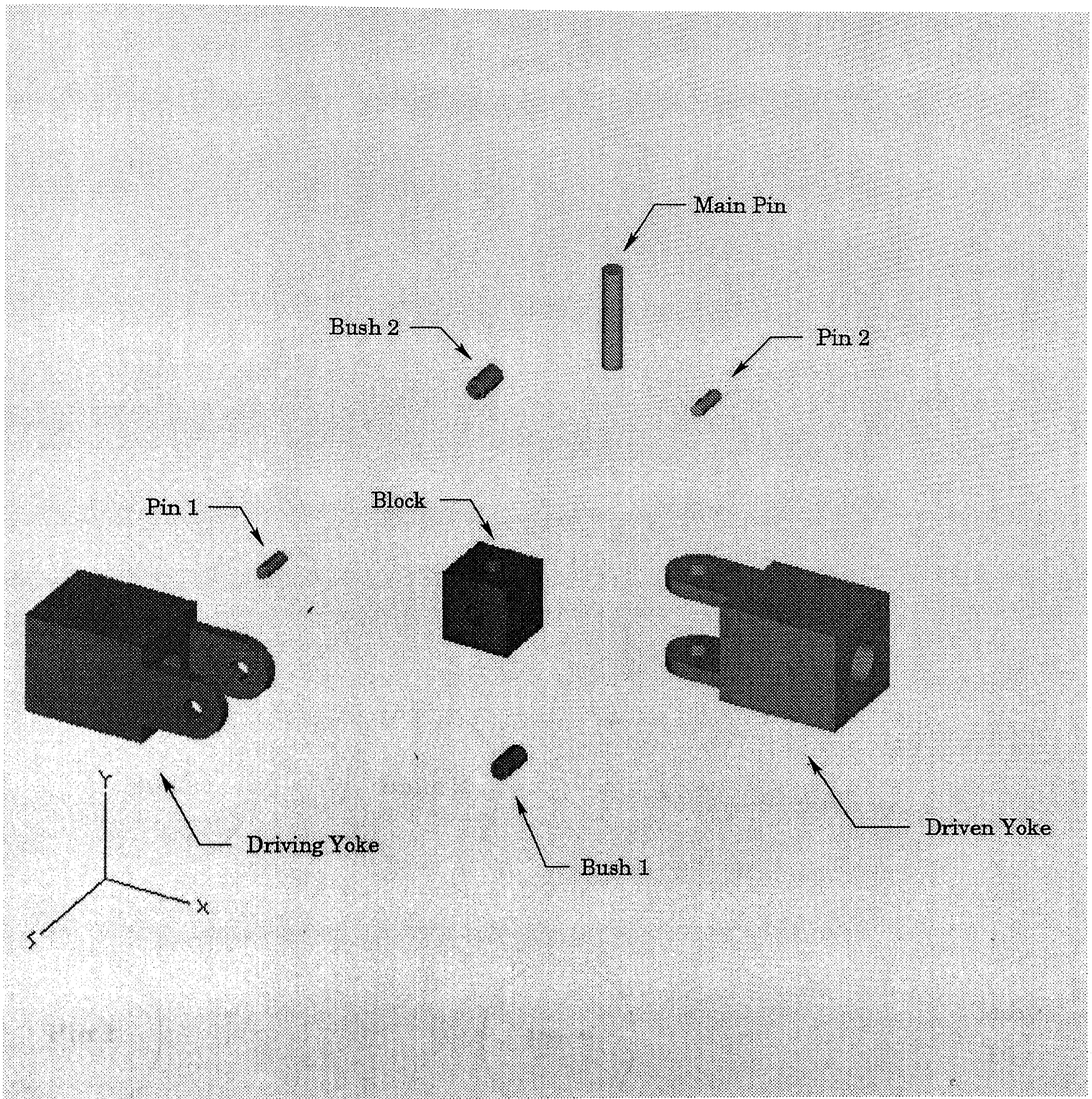

Figure 6.1: Disassembly of Cardan Universal Joint

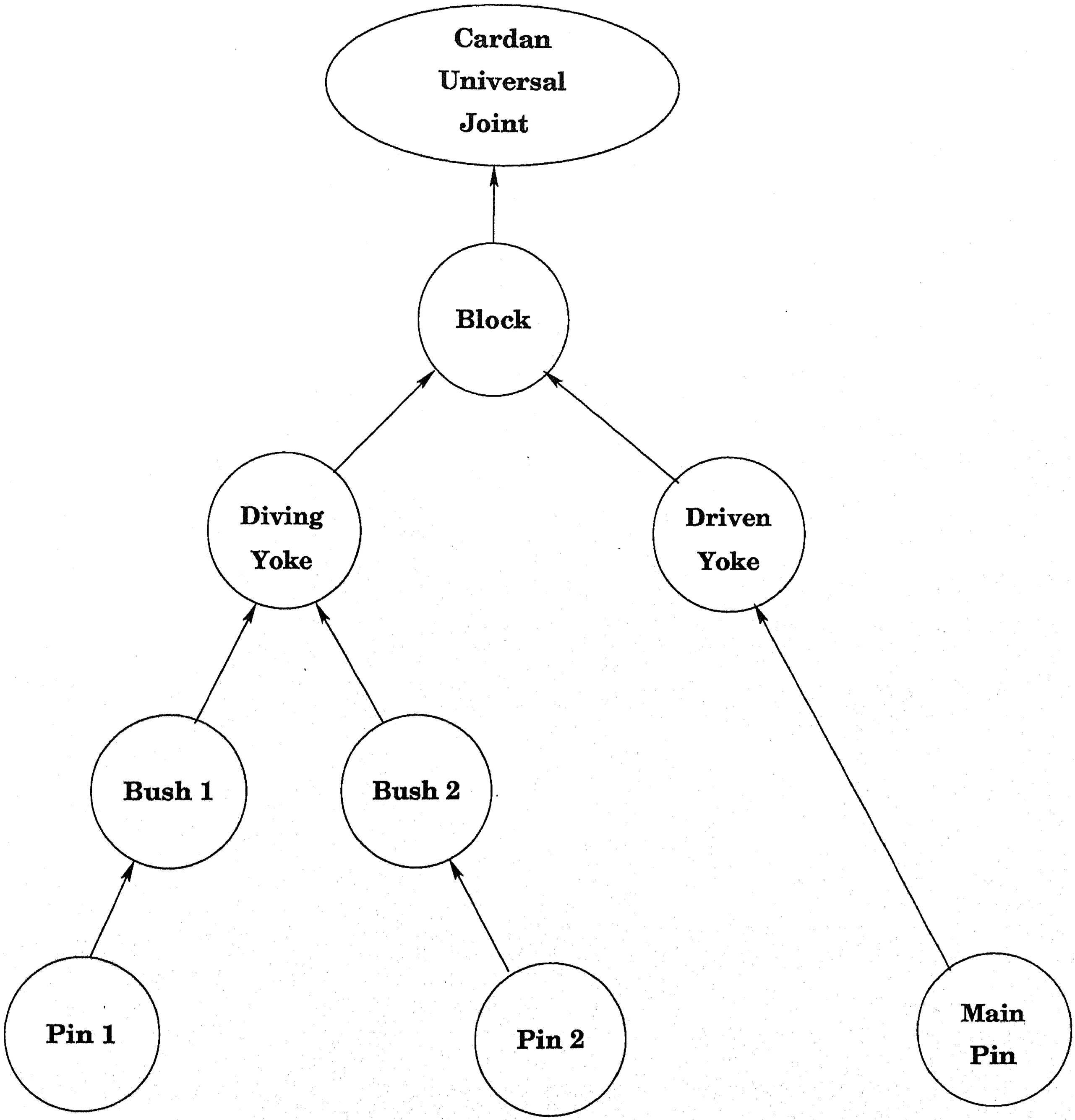

Figure 6.2: Assembly tree of Cardan Universal Joint

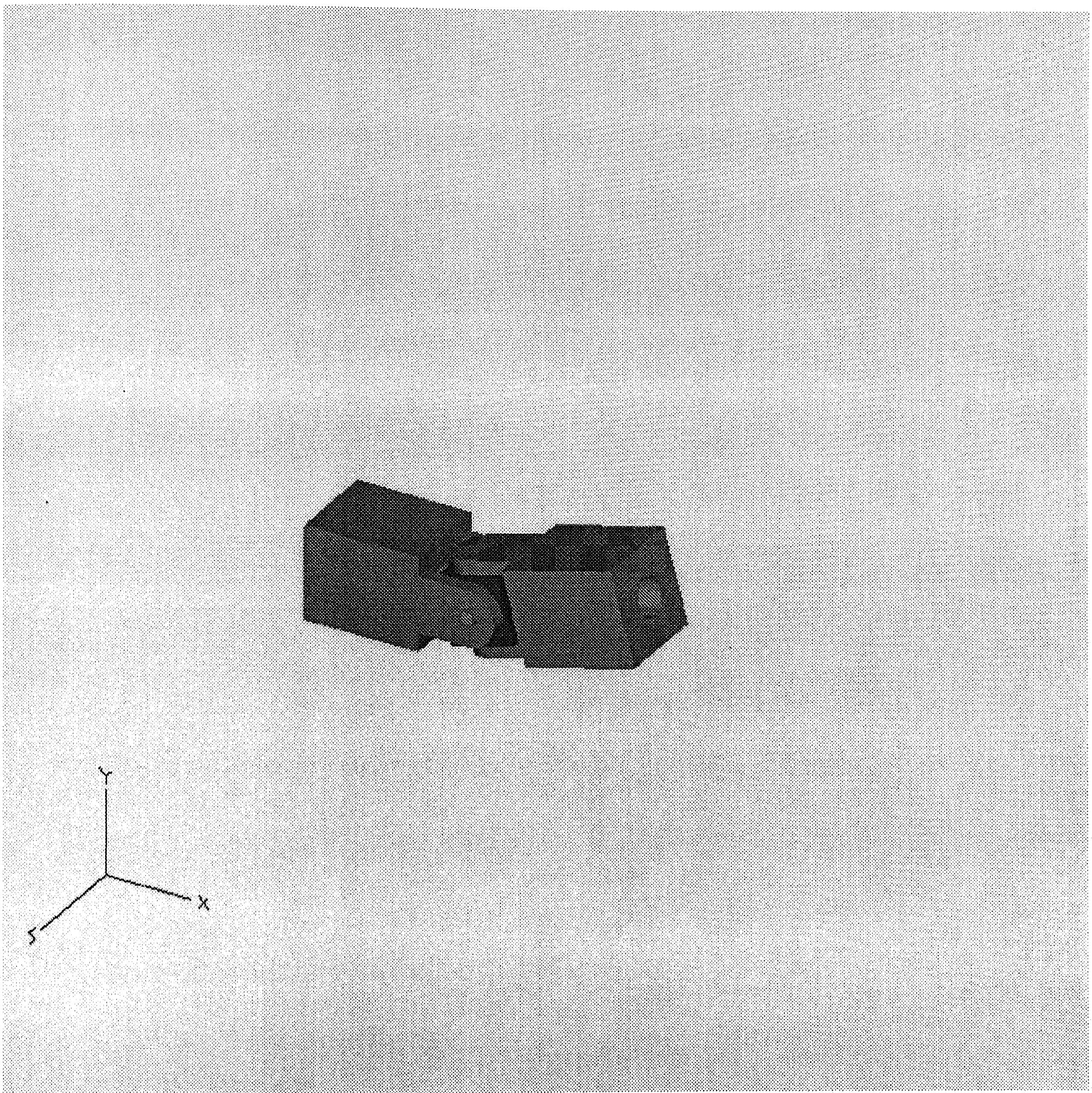

Figure 6.3: Assembly of Cardan Universal Joint

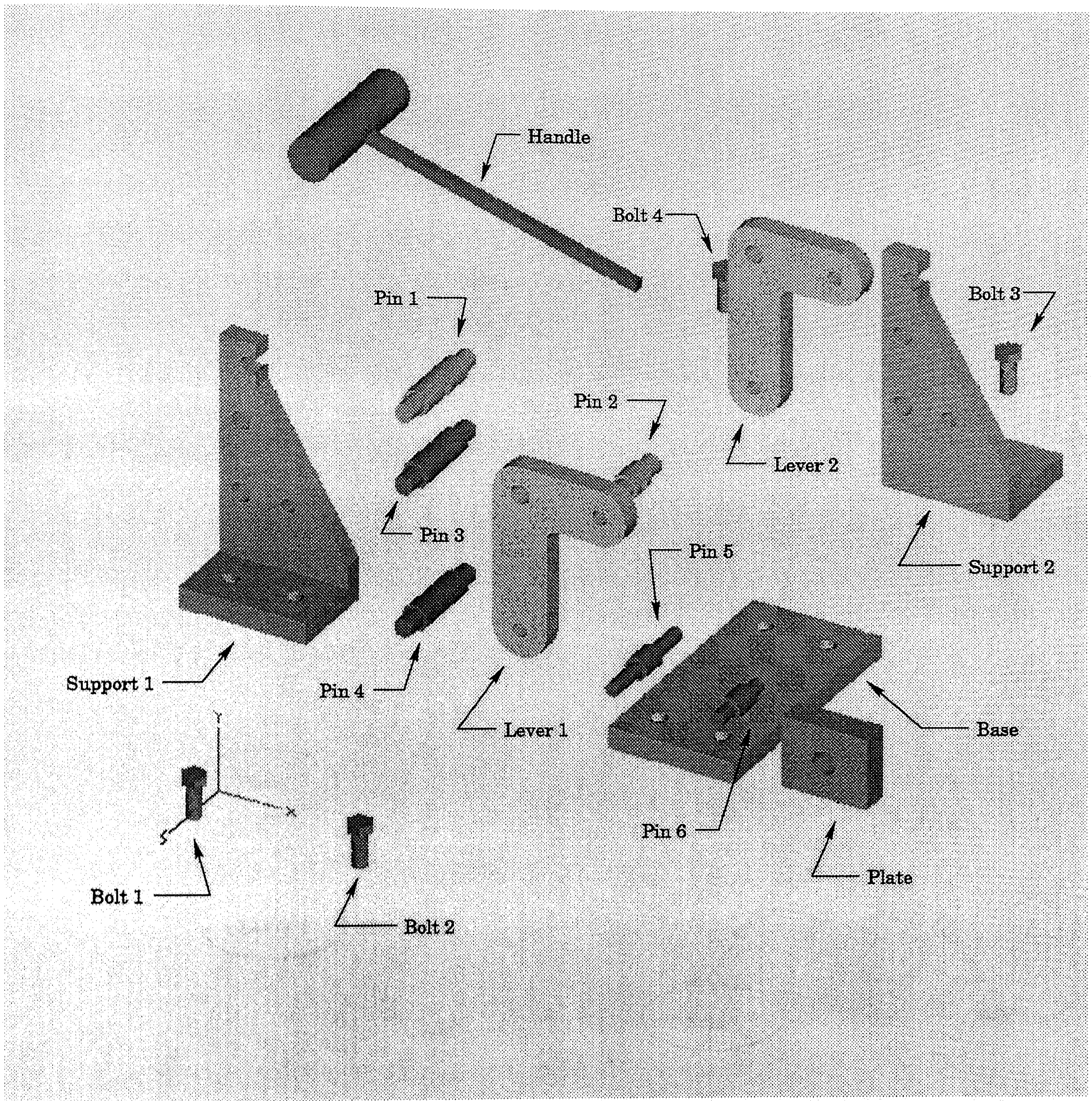

Figure 6.4: Disassembly of Clamp

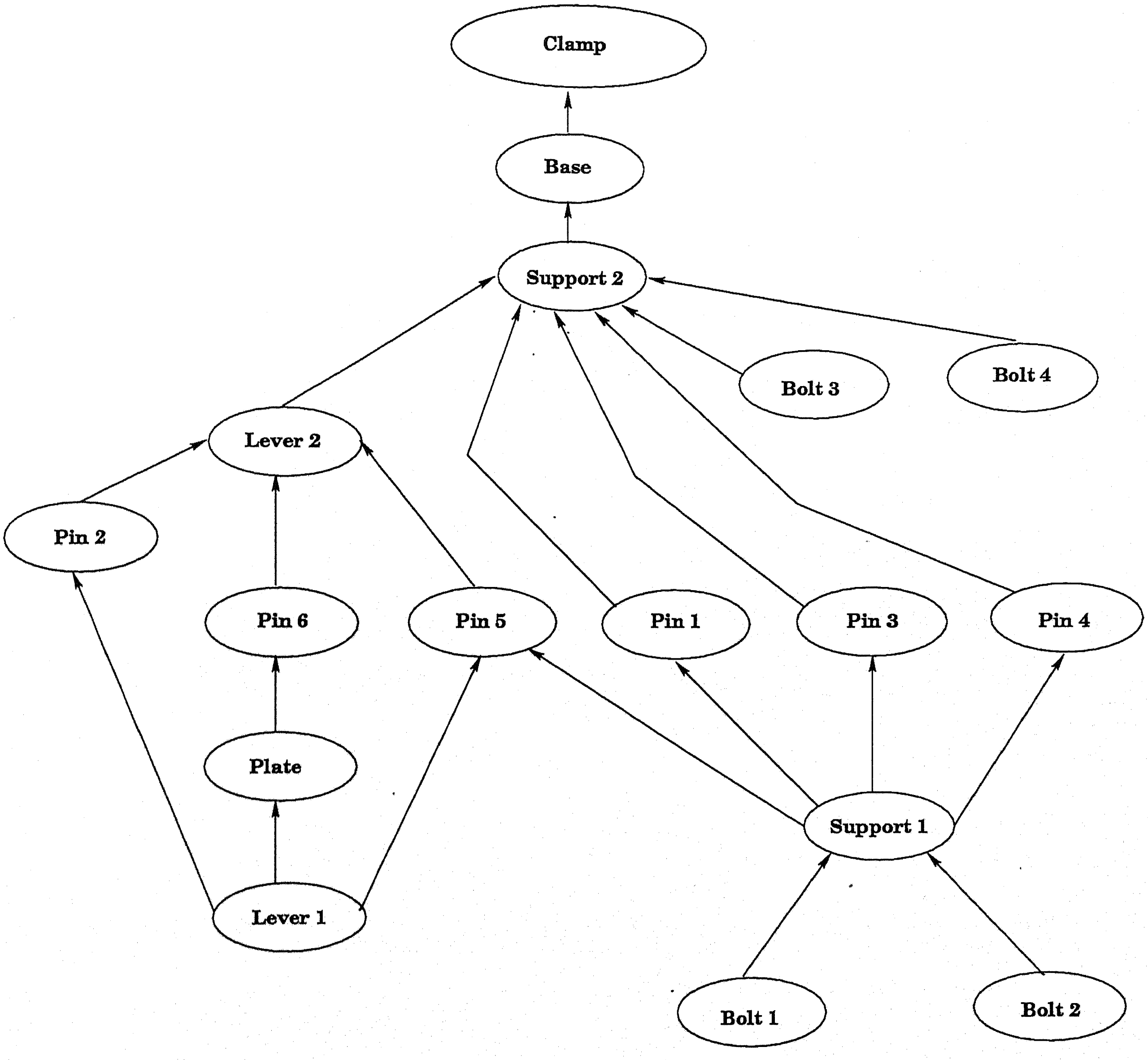

Figure 6.5: Assembly Tree of Clamp

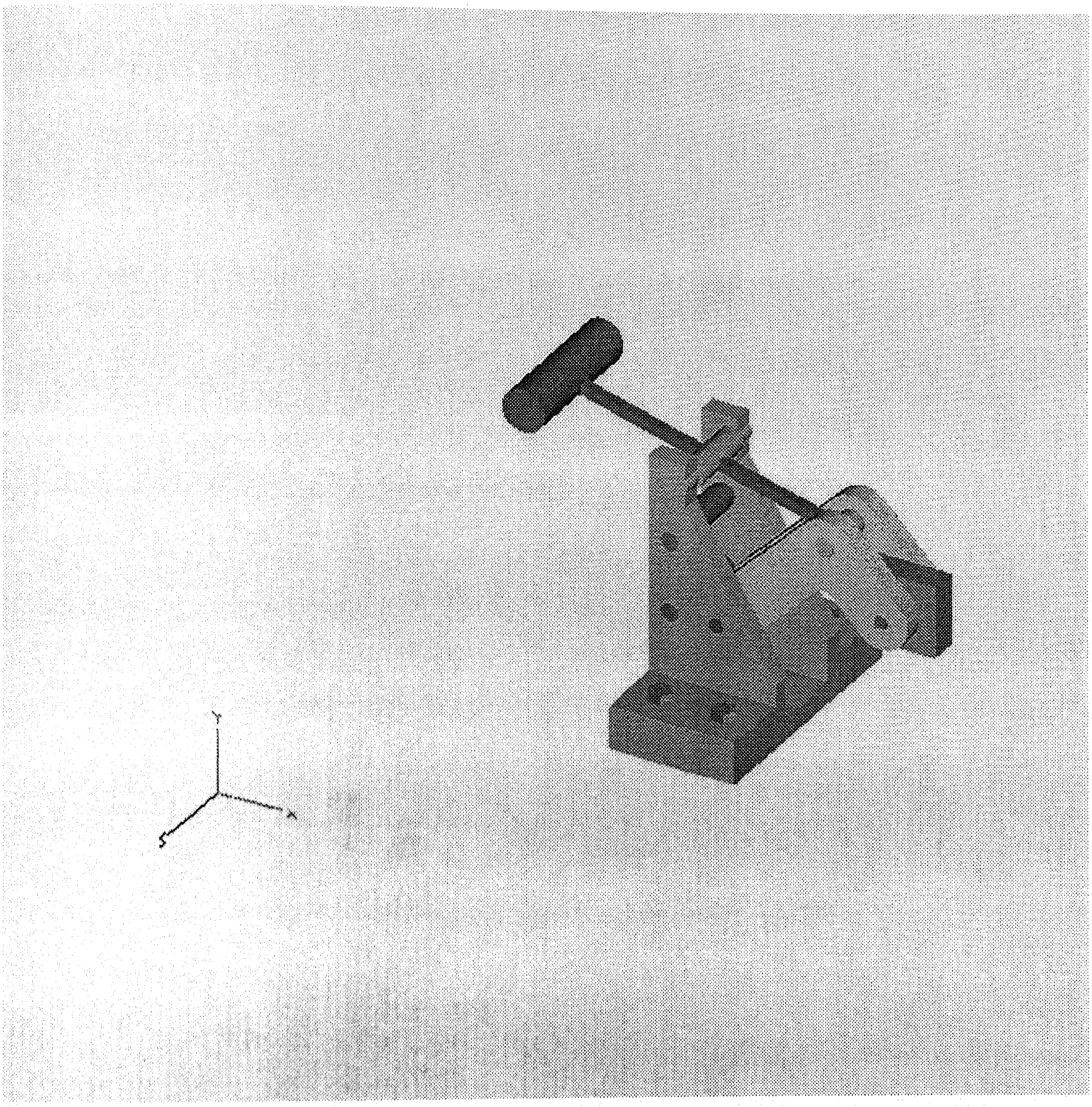

Figure 6.6: Assembly of Clamp

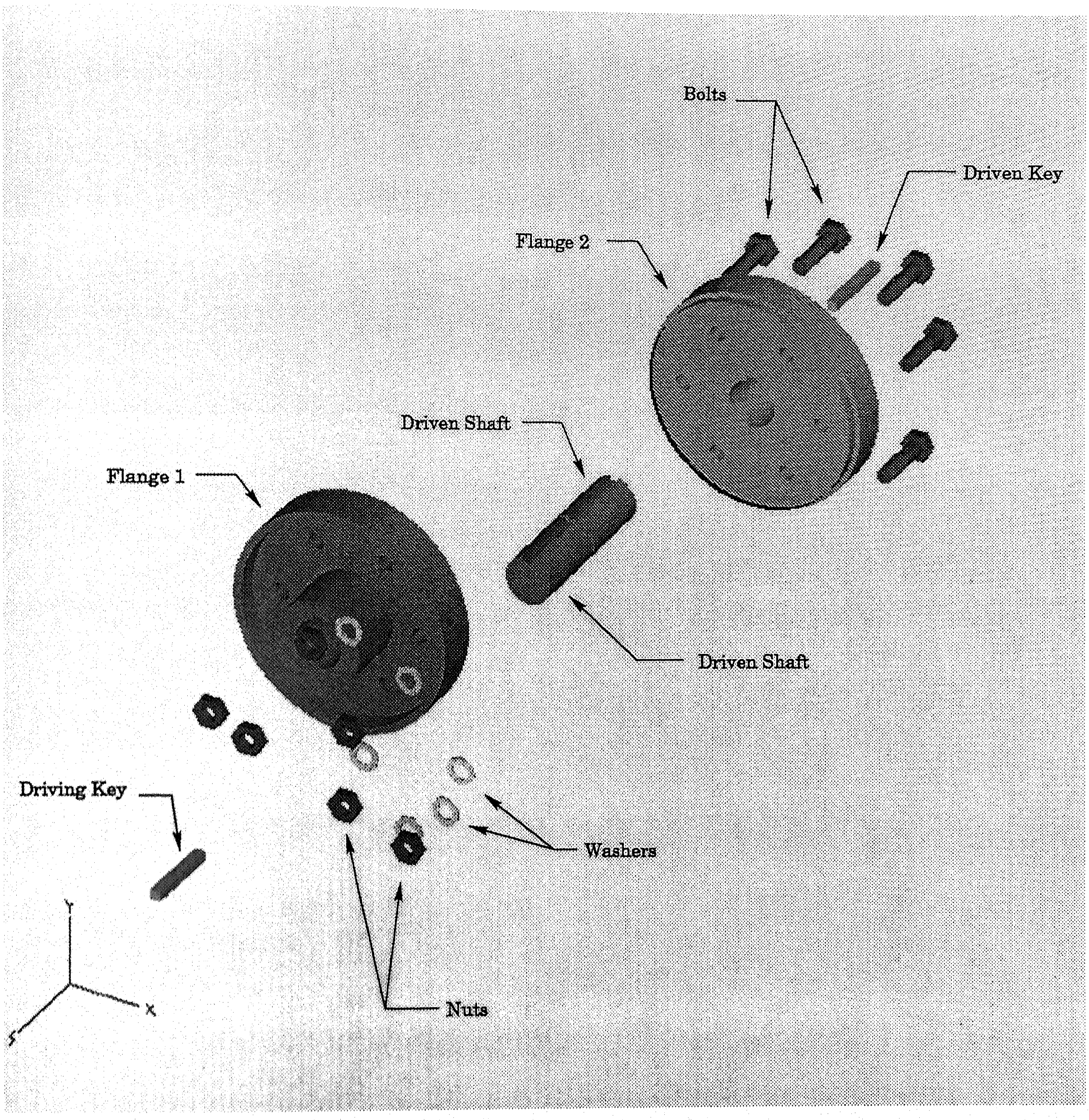

Figure 6.7: Disassembly of Protective Flange Coupling

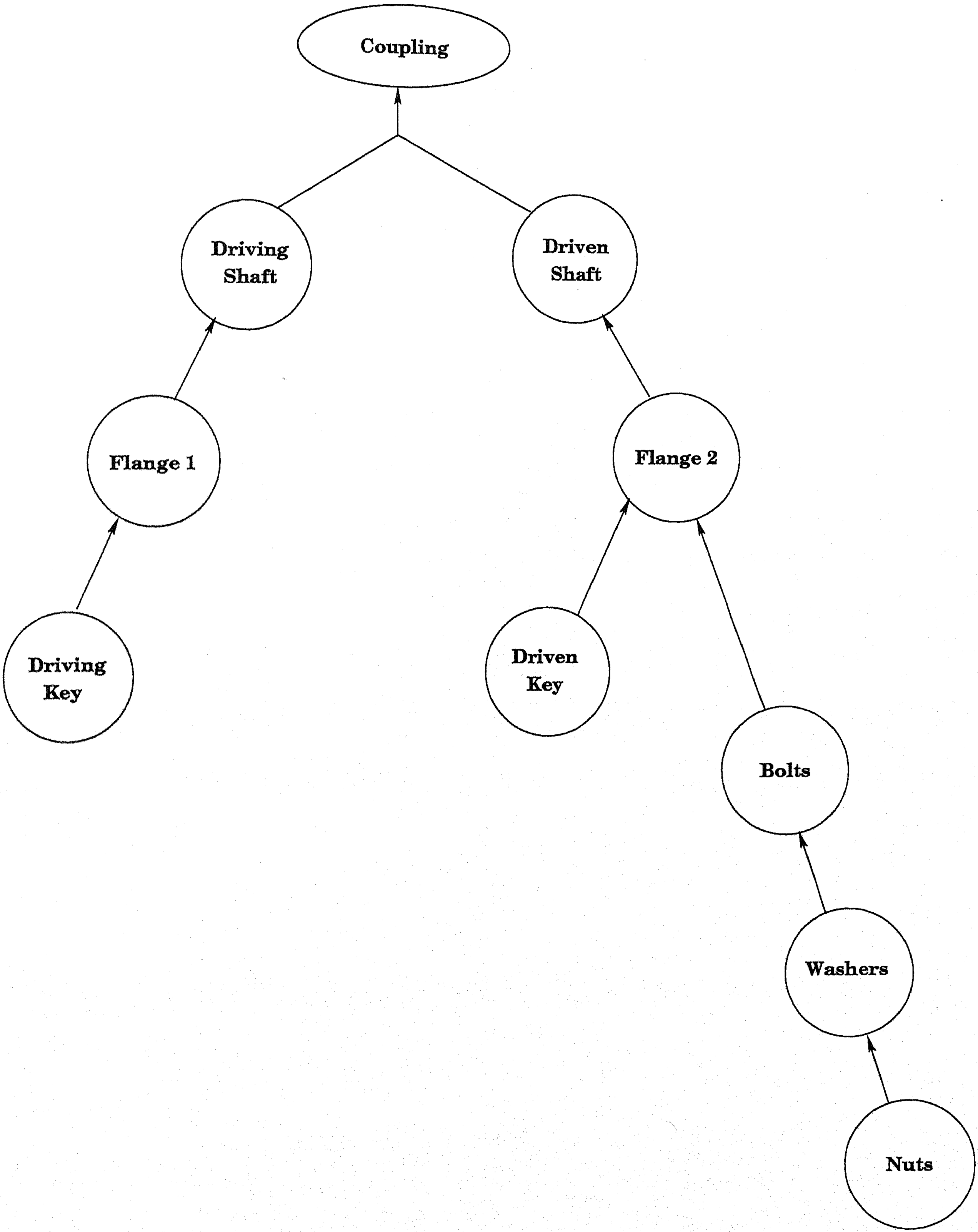

Figure 6.8: Assembly tree of Protective Flange Coupling

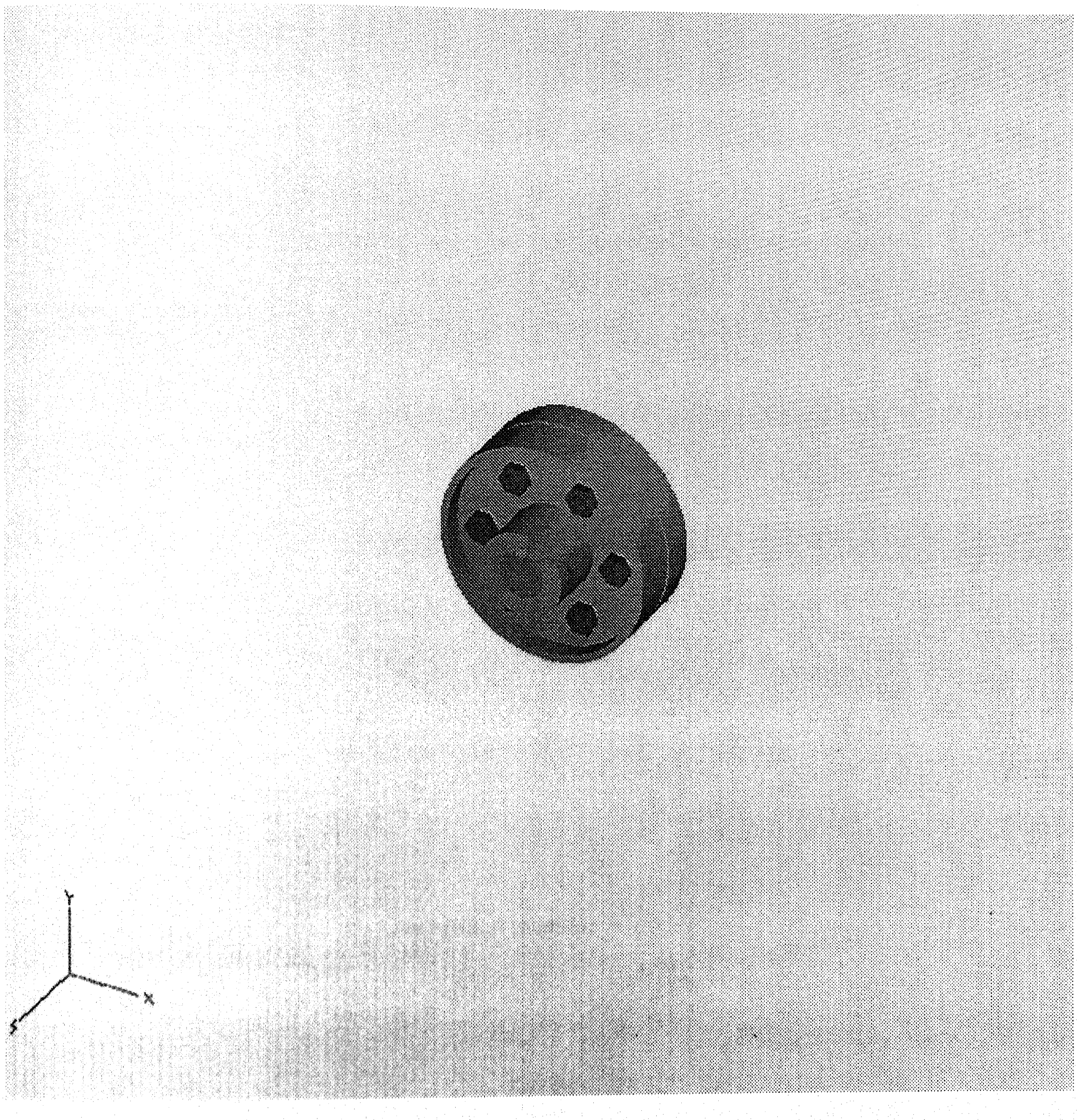

Figure 6.9: Assembly of Protective Flange Coupling

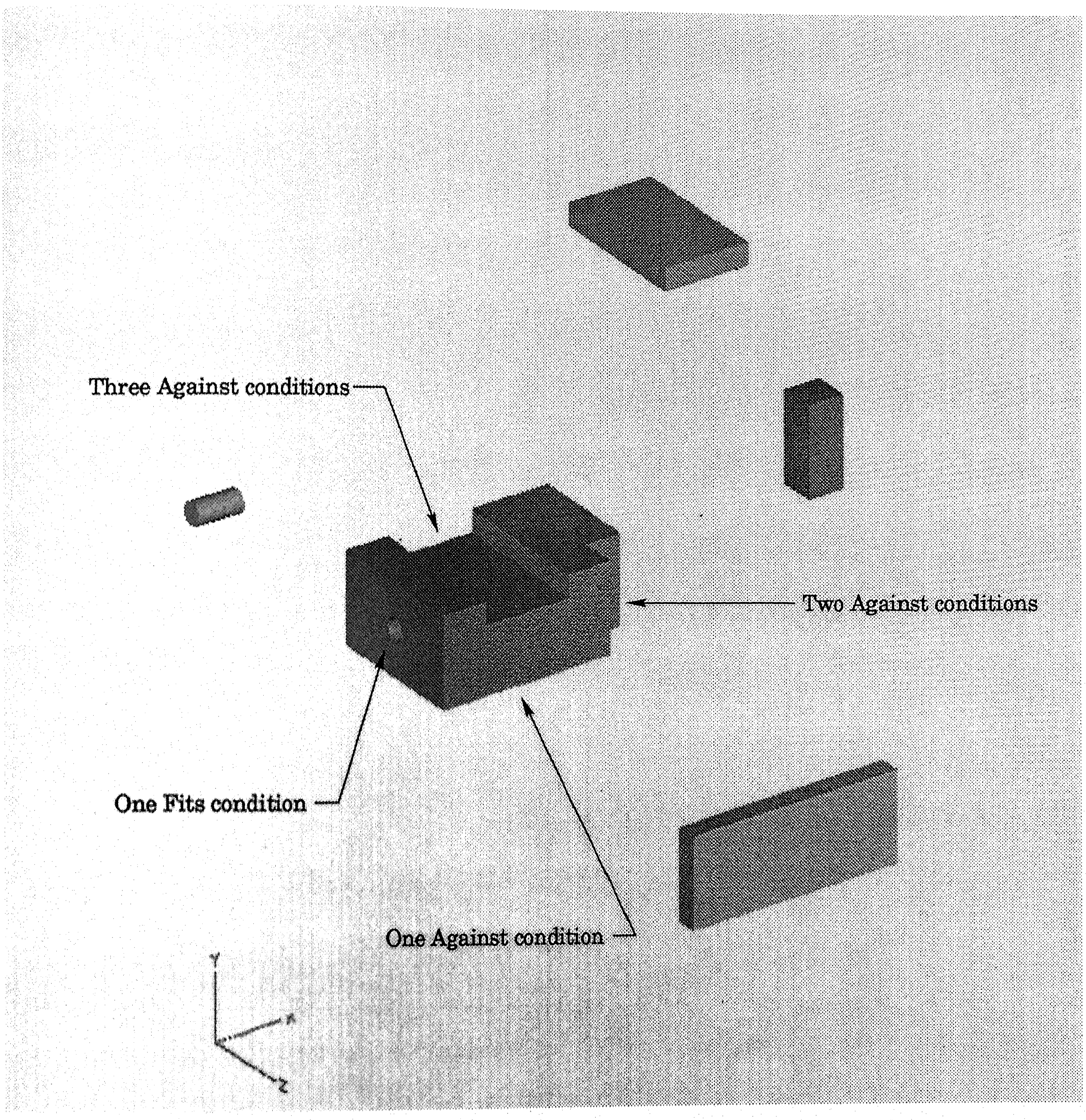

Figure 6.10: Disassembly: Problem of Different Mating Conditions

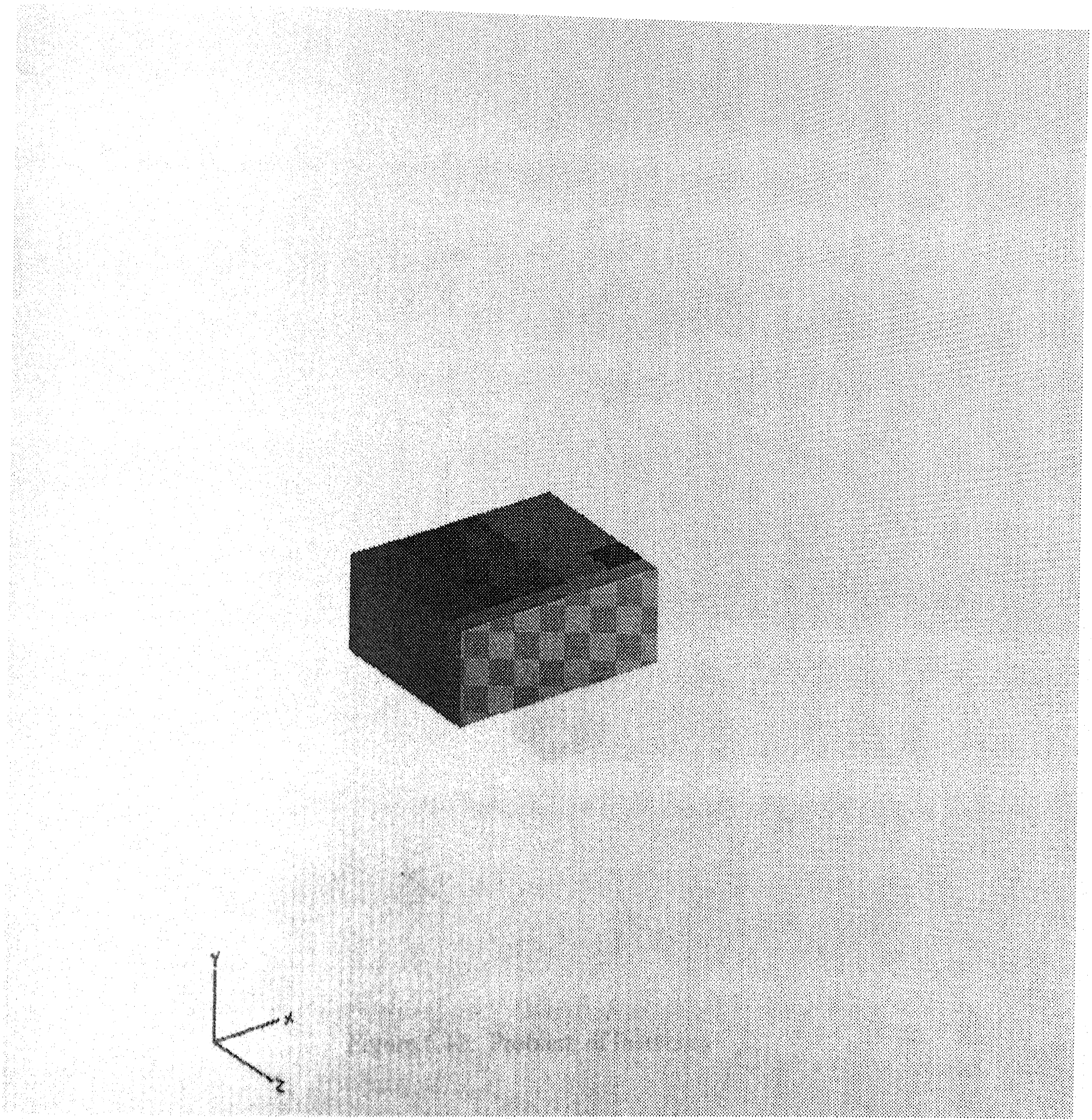

Figure 6.11: Assembly: Problem of Different Mating Conditions

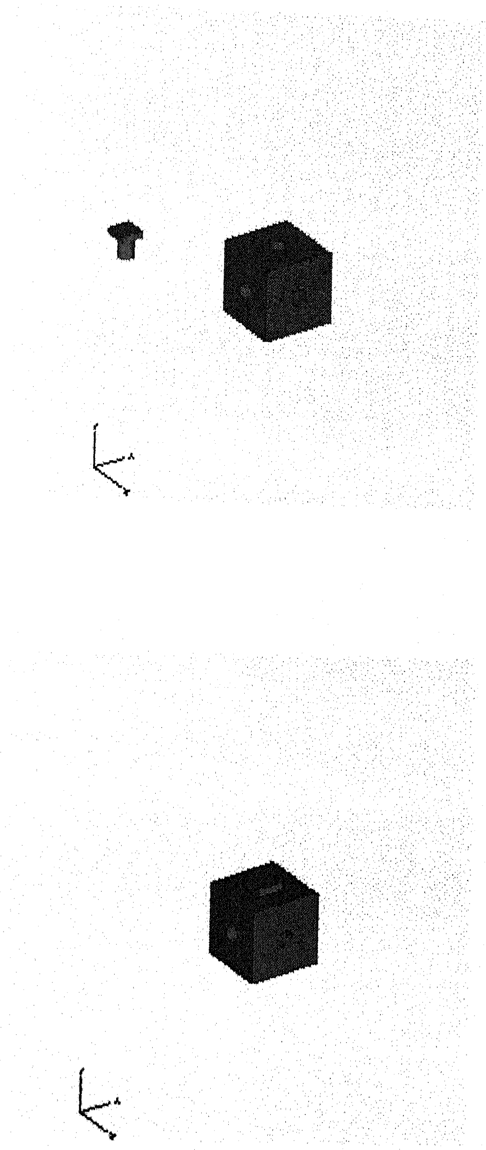

Figure 6.12: Problem of Instances

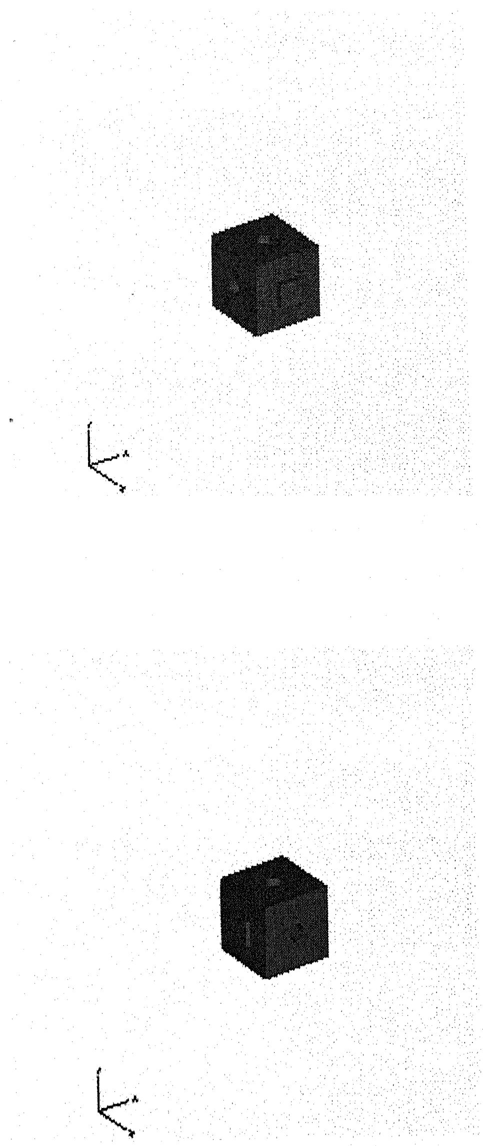

Figure 6.13: Problem of Instances

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This project began with the aim of providing a simple and user friendly assembly modeling system for all mechanical parts. This is achieved using two algorithms as we discussed earlier. These two algorithms were successfully tested for performing different mechanical assemblies with different mating conditions. Although both the algorithms require the user to input the necessary geometric information, each of them has its own advantages. We will discuss them in brief

- In Homogeneous Transformation Matrix algorithm, user has to input the initial and final positions of the various parts in the assembly and then algorithm evaluates the proper translations. This is the simplest way of representing an assembly. Where as, in Mating Conditions algorithm besides giving the necessary geometric information, user has to derive the Jacobian matrix from mating conditions in hand. This makes the algorithm relatively clumsy.

- The Homogeneous Transformation Matrix algorithm can only find out the translations and not the required rotations. So its use is limited for the cases where all the parts have local coordinate system parallel to the global coordinate system of the base part. Whereas, the Mating Conditions algorithm is capable of calculating the orientations of the various parts in the assembly if their coordinates are parallel to one plane and inclined to the remaining two planes.

- Both the algorithms can be used to perform the assembly from given set of components. But one has to select the approach depending on the type of mating conditions involved in the assembly. If assembly has the parts with orientation same as that of the base part, Homogeneous Transformation Matrix algorithm suffices. Whereas, if the assembly has few parts whose orientations are not same as that of the base part, one has to go for the Mating Conditions algorithm.

- Through the employment of the interactive assembling the designer may concentrate more time upon the analysis and performance of an assembly once this analysis has been added to this work. He will also be less apprehensive about considering other assembly alternatives knowing that such a tool is at his disposal.

7.2 Scope for Further Work

Although this work is completed only up to the assembly modeling stage, one can extend it beyond this. Few of these possible extensions are discussed below:

- **Analysis:**

  Among the popular analysis tools are the cross-sectional views of the assemblies, mass property calculations, inference checking, finite element analysis. Thus extending this approach the user can analyse individual parts separately or group of them.

- **Simulation:**

  Similarly, one can simulate the assembly to perform kinematic and dynamic analysis on it and evaluate the performance.

- **Automatic Assembly Sequence Generation:**

  Since, we have used the mating conditions for inferring positions of the parts in the assembly, we can develop an algorithm that will give us different possible sequences from mating conditions. Then it would be very easy to find out the most optimal sequence.

- **Computer-controlled robots:**

  Finally, this work may be linked to the task-level programming of a robot for the assembly task which uses a coding system to generate a program for manipulating computer controlled robots. A coding system consists of many general procedures. The system is given component descriptions necessary for initiating the general procedure (e.g. coordinate of the component, grasp point etc.) according to the sequence in which the components are assembled. So the sequence of instantiated procedures forms an assembly instruction program to control the robot.

References

1. Ko, H., and L. Lee: "Automatic Assembling Procedure Generation from Mating Conditions," *CAD J.*, vol. 19, no. 1, pp. 3-10, 1987.

2. Lee, K., and G. Andrews: "Inference of the Positions of Components in an Assembly: Part 2," *CAD J.*, vol. 17, no. 1, pp. 20-24, 1985.

3. Lee, K., and D. C. Gossard: "A Hierarchical Data Structure for Representing Assemblies: Part 1," *CAD J.*, vol. 17, no. 1, pp. 15-19, 1985.

4. Rocheleau, D. N., and K. Lee: "Systems for Interactive Assembly Modeling," *CAD J.*, vol. 19, no. 2, pp. 65-72, 1987.

5. Wesley, M. A., T. Lozano-Perez, L. I. Liberman, and D. D. Grossman: "A Geometric Modeling System for Automated Mechanical Assembly," *IBM J., Res.Dev.*, vol. 24, no. 1, pp. 64-74, 1980.

6. Kanai, S., Takahashi, H., and Makino, H., 1996. Aspen: Computer Aided Assembly Sequence Planning and Evaluation System Based on Predetermined Time Standard. *Annals of the CIRP*, 45, 1 : 35-39.

7. Latombe, J., Wilson, R. H., and Cazals, F., 1997. Assembly Sequence with Toleranced Parts. *Computer-Aided Design*, 29, 2 : 159-174.

Appendix (A)
Functions used for Code

```
/*** Function for Matrix Multiplication ***/
GLfloat matmul( GLint x, GLint r, GLint y, GLfloat *p, GLfloat *q, GLfloat *z )
{
        GLint i, j, m, n, k;
        GLfloat prod, w1[15][15], w2[15][15], w3[15][15];

        for(i = 0, j = 0; i < x, j < r; i++, j++)
        {
        w1[i][j] = 0.0;
        w1[i][j] = *p;
        *p++;
        }

        for(i = 0, j = 0; i < r, j < y; i++, j++)
        {
        w2[i][j] = 0.0;
        w2[i][j] = *q;
        *q++;
        }

        for(m = 0, n = 0; m < x, n < y; m++, n++)
        {
        prod = 0;
        for(k = 0; k < r; k++)
        {
        w3[i][j] = 0.0;
        prod = prod + w1[m][k] * w2[k][n];
        w3[m][n] = prod;
        }
        }

        for(m = 0, n = 0; m < x, n < y; m++, n++)
        {
        *z = w3[m][n];
        *z++;
```

```
        }
}
/*** Function for Matrix Inversion ***/
GLfloat matinv( GLint x, GLint y, GLfloat *w, GLfloat *s )
{
        GLfloat l[10][10], m[10][15], n[10][15], a[10][15];
        GLint i, j, k;

        for(i = 0, j = 0; i < x, j < y; i++, j++)
        {
        l[i][j] = 0.0;
        l[i][j] = *w;
        *w++;
        }
/*** Procedure for augmented matrix ***/
        for(i = 0, j = 0; i < x, j < (y + x); i++, j++)
        {
        a[i][j] = 0.0;
        if(j >= x)
        a[i][i + x] = 1.0;
        else
        a[i][j] = 0.0;
        }

        for(i = 0, j = 0; i < x, j < x; i++, j++)
        a[i][j] = l[i][j];

        for(i = 0, j = 0; i < x, j < (y + x); i++, j++)
        {
        n[i][j] = 0.0;
        m[i][j] = 0.0;
        m[i][j] = a[i][j];
        }
/*** Does matrix inversion by G-J method ***/
        for(k = 0, i = 0; k < x, i < x; k++, i++)
        {
        for(j = 0; j < (y + x); j++)
        {
```

```
if(k > 0)
m[i][j] = n[i][j];
if(i == k)
n[i][j] = m[i][j]/m[i][k];
else
n[i][j] = m[i][j];
}

for(i = 0, j = 0; i < x, j < (y + x); i++, j++)
{
if(i != k)
n[i][j] = m[i][j] - m[i][k] * n[k][j];
}
}

for(i = 0, j = y; i < x, j < (y + x); i++, j++)
{
*s = n[i][j];
*s++;
}
}
```

Appendix (B)
Code for Homogeneous Transformation Matrix Approach

```
/*** Assigns the rotational matrix for α, β and γ values ***/
GLfloat assmat( GLfloat xa, GLfloat ya, GLfloat za, GLfloat *tmt )
{
        GLfloat ax, ay, az, al, bt, gm, tmat[4][4];
        ax = xa;    ay = ya;    az = za;

        al = (az * PI)/180.0;
        bt = (ay * PI)/180.0;
        gm = (ax * PI)/180.0;

        tmat[0][0] = cos(al) * cos(bt);
        tmat[0][1] = -sin(al) * cos(bt);
        tmat[0][2] = sin(bt);
        tmat[0][3] = 0.0;

        tmat[1][0] = sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm);
        tmat[1][1] = cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm);
        tmat[1][2] = -cos(bt) * sin(gm);
        tmat[1][3] = 0.0;

        tmat[2][0] = sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm);
        tmat[2][1] = cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm);
        tmat[2][2] = cos(bt) * cos(gm);
        tmat[2][3] = 0.0;

        tmat[3][0] = 0.0;
        tmat[3][1] = 0.0;
        tmat[3][2] = 0.0;
        tmat[3][3] = 1.0;

        for(i = 0, j = 0; i < 4, j < 4; i++, j++)
        {
        *tmt = tmat[i][j];
        *tmt++;
```

```
        }
}
/*** Calculates the x, y and z components of translation when the normals of the mating
        faces are already opposite ***/
GLfloat calxyz( GLfloat *c1, GLfloat *c3, GLfloat *x, GLfloat *y, GLfloat *z )
{
        GLfloat p1[4][1], p3[4][1];
        GLfloat tmat[4][4], mtmat[4][4], pro[4][1], mpro[4][1], fcod[4][1];
        GLfloat rm[4][1] = {1.0, 1.0, 1.0, 0.0};

        for(i = 0, j = 0; i < 4, j < 1; i++, j++)
        {
        p1[i][j] = *c1;     *c1++;
        p3[i][j] = *c3;     *c3++;
        }

        /*** Calling the rotational matrix assignment function ***/
        assmat(0.0, 0.0, 0.0, &tmat[0][0]);

        for(i = 0, j = 0; i < 4, j < 4; i++, j++)
        {
        if(j > 2)
        mtmat[i][j]=1.0;
        else
        mtmat[i][j] = tmat[i][j];
        }

        /*** Calling the matrix multiplication function ***/
        matmul(4, 4, 1, &mtmat[0][0], &p3[0][0], &pro[0][0]);

        for(i = 0, j = 0; i < 4, j < 1; i++, j++)
        {
        mpro[i][j] = pro[i][j] - rm[i][j];
        fcod[i][j] = p1[i][j] - mpro[i][j];
        }

        *x = fcod[0][0];     *y = fcod[1][0];     *z = fcod[2][0];
    }
```

```
/*** Calculates the x, y and z components of translation when the normals of the mating
        faces are not opposite ***/
GLfloat calxyz( GLfloat *c1, GLfloat *c3, GLfloat *ct, GLfloat xa, GLfloat ya, GLfloat za,
                GLfloat *x, GLfloat *y, GLfloat *z )
{
        GLfloat p1[4][1], p3[4][1];
        GLfloat tmat[4][4], mtmat[4][4], pro[4][1], mpro[4][1], fcod[4][1];
        GLfloat p33[4][1], p3t[4][1], mt[4][4], np3[4][1];
        GLfloat rm[4][1] = {1.0, 1.0, 1.0, 0.0};

        for(i = 0; i < 4; i++)
        for(j = 0; j < 1; j++)
        {
        p1[i][j] = *c1;     *c1++;
        p3[i][j] = *c3;     *c3++;
        p3t[i][j] = *ct;    *ct++;
        }

        /*** Translating the initial position vector towards origin ***/
        for(i = 0; i < 4; i++)
        for(j = 0; j < 1; j++)
        p33[i][j] = p3[i][j] - p3t[i][j];

        /*** Finding the coordinates of the initial position vector
                after corresponding rotation about origin ***/
        assmat(xa, ya, za, &mt[0][0]);
        matmul(4, 4, 1, &mt[0][0], &p33[0][0], &np33[0][0]);

        /*** Translating the initial position vector back to its original position ***/
        for(i = 0; i < 4; i++)
        for(j = 0; j < 1; j++)
        np33[i][j] = np33[i][j] + p3t[i][j];

        /*** Calling the rotational matrix assignment function ***/
        assmat(0.0, 0.0, 0.0, &tmat[0][0]);
```

```
for(i = 0; i < 4; i++)
for(j = 0; j < 4; j++)
{
if(j > 2)
mtmat[i][j] = 1.0;
else
mtmat[i][j] = tmat[i][j];
}

/*** Calling the matrix multiplication function ***/
matmul(4, 4, 1, &mtmat[0][0], &np33[0][0], &pro[0][0]);

for(i = 0; i < 4; i++)
for(j = 0; j < 1; j++)
{
mpro[i][j] = pro[i][j] - rm[i][j];
fcod[i][j] = p1[i][j] - mpro[i][j];
}

*x = fcod[0][0];    *y = fcod[1][0];    *z = fcod[2][0];
}
```

Appendix (C)
Code for Mating Conditions Approach

/$**$ *Function for calculating* α, β, γ, x , y *and* z *variables using Mating Conditions Approach* $**$/

GLfloat calxyz(GLint dir, GLint s, GLfloat $*p_{11}$, GLfloat $*p_{22}$, GLfloat $*p_{33}$, GLfloat $*p_{44}$, GLfloat $*t_x$, GLfloat $*t_y$, GLfloat $*t_z$)

{

/$**$ *Variable Declaration* $**$/

GLfloat $jac[9][6], trjac[6][9], protrjac[6][6], invprotrjac[6][6]$;

GLfloat $slpro[6][9], fpro[6][1], resv[9][1]$;

GLfloat $x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4$;

GLfloat $p_1[3], p_2[3], p_3[3], p_4[3]$;

GLfloat $gamma, beta, alpha, gm, bt, al, x, y, z$;

for($i = 0; i < 3; i++$)

{

$p_1[i] = *p_{11}; *p_{11}++$;

$p_2[i] = *p_{22}; *p_{22}++$;

$p_3[i] = *p_{33}; *p_{33}++$;

$p_4[i] = *p_{44}; *p_{44}++$;

}

$x_1 = p_1[0]; y_1 = p_1[1]; z_1 = p_1[2]$;

$x_2 = p_2[0]; y_2 = p_2[1]; z_2 = p_2[2]$;

if($s == 1$)

{

$x_3 = p_3[0]; y_3 = p_3[1]; z_3 = p_3[2]$;

$x_4 = p_4[0]; y_4 = p_4[1]; z_4 = p_4[2]$;

}

else

{

$x_3 = p_4[0]; y_3 = p_4[1]; z_3 = p_4[2]$;

$x_4 = p_3[0]; y_4 = p_3[1]; z_4 = p_3[2]$;

}

$x = 0.0; y = 0.0; z = 0.0$;

$gamma = 0.0; beta = 0.0; alpha = 0.0$;

$gm = (PI * gamma)/180.0$;

```
        bt = (PI * beta)/180.0;
        al = (PI * alpha)/180.0;
switch(dir)
{
        case 1:         /*** Parallel to X axis ***/
        jac[0][0] = -sin(al) * cos(bt);         /*** Jacobian Matrix ***/
        jac[0][1] = -cos(al) * sin(bt);
        jac[0][2] = 0.0;
        jac[0][3] = 0.0;
        jac[0][4] = 0.0;
        jac[0][5] = 0.0;

        jac[1][0] = (cos(al) * cos(gm)) - (sin(al) * sin(bt) * sin(gm));
        jac[1][1] = cos(al) * cos(bt) * sin(gm);
        jac[1][2] = -(sin(al) * sin(gm)) + (cos(al) * sin(bt) * cos(gm));
        jac[1][3] = 0.0;
        jac[1][4] = 0.0;
        jac[1][5] = 0.0;

        jac[2][0] = (cos(al) * sin(gm)) + (sin(al) * sin(bt) * cos(gm));
        jac[2][1] = -cos(al) * cos(bt) * cos(gm);
        jac[2][2] = (sin(al) * cos(gm)) + (cos(al) * sin(bt) * sin(gm));
        jac[2][3] = 0.0;
        jac[2][4] = 0.0;
        jac[2][5] = 0.0;

        jac[3][0] = -(2.0 * x3 * cos(al) * sin(al) * cos(bt) * cos(bt))+
        (y3 * sin(al) * sin(al) * cos(bt) * cos(bt))- (y3 * cos(al) * cos(al) * cos(bt) * cos(bt))-
        (z3 * sin(bt) * sin(al) * cos(bt)) - (x * sin(al) * cos(bt))+
        (x1 * sin(al) * cos(bt)) + (2.0 * x3 * sin(al) * cos(al) * cos(gm) * cos(gm))-
        (y3 * cos(al) * cos(al) * cos(gm) * cos(gm))-
        (y3 * sin(al) * sin(al) * cos(gm) * cos(gm))+
        (2.0 * z3 * cos(bt) * sin(gm) * cos(al) * cos(gm))+
        (y * cos(al) * cos(gm)) - (y1 * cos(al) * cos(gm))-
        (2.0 * x3 * cos(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm))+
        (y3 * sin(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm))-
        (y3 * cos(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm))-
        (z3 * cos(bt) * sin(bt) * sin(al) * sin(gm) * sin(gm))-
```

$(y * sin(al) * sin(bt) * sin(gm)+$
$(y_1 * sin(al) * sin(bt) * sin(gm))$
$+(2.0 * x_3 * sin(al) * cos(al) * sin(gm) * sin(gm))-$
$(y_3 * sin(al) * sin(al) * sin(gm) * sin(gm))+$
$(y_3 * cos(al) * cos(al) * sin(gm) * sin(gm))+$
$(z * cos(al) * sin(gm))- (z_1 * cos(al) * sin(gm))-$
$(2.0 * x_3 * cos(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))+$
$(y_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))-$
$(y_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))+$
$(z_3 * cos(bt) * sin(al) * sin(bt) * cos(gm) * cos(gm))+$
$(z * sin(al) * sin(bt) * cos(gm)) - (z_1 * sin(al) * sin(bt) * cos(gm));$

jac[3][1] $= -(2.0 * x_3 * cos(al) * cos(al) * cos(bt) * sin(bt))+$
$(2.0 * y_3 * sin(al) * cos(al) * cos(bt) * sin(bt))-$
$(z_3 * sin(bt) * sin(bt) * cos(al))+$
$(z_3 * cos(bt) * cos(bt) * cos(al)) - (x * cos(al) * sin(bt))+$
$(x_1 * cos(al) * sin(bt)) - (2.0 * z_3 * sin(bt) * sin(gm) * sin(al) * cos(gm))+$
$(2.0 * x_3 * cos(al) * cos(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm))-$
$(2.0 * y_3 * cos(al) * sin(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm))+$
$(z_3 * cos(bt) * cos(bt) * cos(al) * sin(gm) * sin(gm))-$
$(z_3 * sin(bt) * sin(bt) * cos(al) * sin(gm) * sin(gm))+$
$(y * sin(al) * cos(bt) * sin(gm)) - (y_1 * sin(al) * cos(bt) * sin(gm))-$
$(2.0 * x_3 * cos(al) * cos(al) * sin(bt) * cos(bt) * cos(gm) * cos(gm))$
$-(2.0 * y_3 * cos(al) * sin(al) * sin(bt) * cos(bt) * cos(gm) * cos(gm))-$
$(z_3 * cos(bt) * cos(bt) * cos(al) * cos(gm) * cos(gm))+$
$(z_3 * sin(bt) * sin(bt) * cos(al) * cos(gm) * cos(gm))-$
$(z * cos(al) * cos(bt) * cos(gm)) + (z_1 * cos(al) * cos(bt) * cos(gm));$

jac[3][2] $= -(2.0 * x_3 * sin(al) * sin(al) * cos(gm) * sin(gm))$-
$(2.0 * y_3 * sin(al) * cos(al) * cos(gm) * sin(gm))+$
$(2.0 * z_3 * cos(bt) * cos(gm) * cos(gm) * sin(al))-$
$(2.0 * z_3 * cos(bt) * sin(gm) * sin(gm) * sin(al))-$
$(y * sin(al) * sin(gm)) + (y_1 * sin(al) * sin(gm))+$
$(2.0 * x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * cos(gm))$-
$(2.0 * y_3 * cos(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * cos(gm))+$
$(2.0 * z_3 * cos(bt) * sin(bt) * cos(al) * sin(gm) * cos(gm))+$
$(y * cos(al) * sin(bt) * cos(gm)) - (y_1 * cos(al) * sin(bt) * cos(gm))+$
$(2.0 * x_3 * sin(al) * sin(al) * sin(gm) * cos(gm))+$

$(2.0 * y_3 * cos(al) * sin(al) * sin(gm) * cos(gm))+$
$(2.0 * z3 * cos(bt) * cos(al) * sin(bt) * cos(gm) * sin(gm))+$
$(z * sin(al) * cos(gm)) - (z_1 * sin(al) * cos(gm))-$
$(2.0 * x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * sin(gm))+$
$(2.0 * y_3 * cos(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * sin(gm))+$
$(z * cos(al) * sin(bt) * sin(gm)) - (z_1 * cos(al) * sin(bt) * sin(gm));$

jac[3][3] $= cos(al) * cos(bt);$
jac[3][4] $= (sin(al) * cos(gm)) + (cos(al) * sin(bt) * sin(gm));$
jac[3][5] $= (sin(al) * sin(gm)) - (cos(al) * sin(bt) * cos(gm));$

jac[4][0] $= (x_2 - x_1) * (x_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm))$
$+y_3 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)));$
jac[4][1] $= (x_2 - x_1) * (x_3 * (cos(al) * cos(bt) * sin(gm))+$
$y_3 * (-sin(al) * cos(bt) * sin(gm)) + z_3 * sin(bt) * sin(gm));$
jac[4][2] $= (x_2 - x_1) * (x_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm))+$
$y_3 * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm)) - z_3 * cos(bt) * cos(gm));$
jac[4][3] $= 0.0;$
jac[4][4] $= (x_2 - x_1);$
jac[4][5] $= 0.0;$

jac[5][0] $= (x_2 - x_1) * (x_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))+$
$y_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$
jac[5][1] $= (x_2 - x_1) * (x_3 * (-cos(al) * sin(bt) * cos(gm))+$
$y_3 * (sin(al) * cos(bt) * cos(gm)) - z_3 * sin(bt) * cos(gm));$
jac[5][2] $= (x_2 - x_1) * (x_3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+$
$y_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) + z_3 * cos(bt) * sin(gm));$
jac[5][3] $= 0.0;$
jac[5][4] $= 0.0;$
jac[5][5] $= (x_2 - x_1);$

jac[6][0] $= (x_2 - x_1) * (x_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm))$
$+y_4 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)));$
jac[6][1] $= (x_2 - x_1) * (x_4 * (cos(al) * cos(bt) * sin(gm))+$
$y_4 * (-sin(al) * cos(bt) * sin(gm)) + z_4 * sin(bt) * sin(gm));$
jac[6][2] $= (x_2 - x_1) * (x_4 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm))+$
$y_4 * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm)) - z_4 * cos(bt) * cos(gm));$
jac[6][3] $= 0.0;$

jac[6][4] $= (x_2 - x_1);$

jac[6][5] $= 0.0;$

jac[7][0] $= (x_2 - x_1) * (x_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) +$

$y_4 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$

jac[7][1] $= (x_2 - x_1) * (x_4 * (-cos(al) * sin(bt) * cos(gm)) +$

$y_4 * (sin(al) * cos(bt) * cos(gm)) - z_4 * sin(bt) * cos(gm));$

jac[7][2] $= (x_2 - x_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm)) +$

$y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) + z_4 * cos(bt) * sin(gm));$

jac[7][3] $= 0.0;$

jac[7][4] $= 0.0;$

jac[7][5] $= (x_2 - x_1);$

jac[8][0] $= 0.0;$

jac[8][1] $= 0.0;$

jac[8][2] $= 1.0;$

jac[8][3] $= 0.0;$

jac[8][4] $= 0.0;$

jac[8][5] $= 0.0;$

$/ * * * \ Residual\ vector * * * /$

resv[0][0] $= (cos(al) * cos(bt) - 1.0);$

resv[1][0] $= ((cos(al) * sin(bt) * sin(gm)) - (sin(al) * cos(gm)));$

resv[2][0] $= ((sin(al) * sin(gm)) - (cos(al) * sin(bt) * cos(gm)));$

resv[3][0] $= ((x_3 * cos(al) * cos(al) * cos(bt) * cos(bt)) -$

$(y_3 * sin(al) * cos(al) * cos(bt) * cos(bt)) + (z_3 * cos(al) * sin(bt) * cos(bt)) +$

$(x * cos(al) * cos(bt)) - (x_1 * cos(al) * cos(bt)) + (x_3 * sin(al) * sin(al) * cos(gm) * cos(gm)) -$

$(y_3 * cos(al) * sin(al) * cos(gm) * cos(gm)) +$

$(2.0 * z_3 * cos(bt) * sin(gm) * sin(al) * cos(gm)) +$

$(y * sin(al) * cos(gm)) - (y_1 * sin(al) * cos(gm)) +$

$(x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$

$(y_3 * cos(al) * sin(al) * sin(gm) * sin(gm) * sin(bt) * sin(bt)) +$

$(z_3 * cos(bt) * sin(bt) * cos(al) * sin(gm) * sin(gm)) +$

$(y * cos(al) * sin(gm) * sin(bt)) - (y_1 * cos(al) * sin(gm) * sin(bt)) +$

$(x_3 * sin(al) * sin(al) * sin(gm) * sin(gm)) + (y_3 * cos(al) * sin(al) * sin(gm) * sin(gm)) -$

$(z_3 * cos(al) * sin(gm) * cos(bt) * cos(gm)) + (z * sin(al) * sin(gm)) -$

$(z_1 * sin(al) * sin(gm)) +$

$(x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) -$

$(y_3 * cos(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))-$
$(z_3 * cos(bt) * cos(al) * sin(bt) * cos(gm) * cos(gm)) - (z * cos(al) * sin(bt) * cos(gm))+$
$(z_1 * cos(al) * sin(bt) * cos(gm)));$
resv[4][0] = $(x_2 - x_1) * (x_3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+$
$y_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_3 * cos(bt) * sin(gm) + y - y_1);$
resv[5][0] = $(x_2 - x_1) * (x_3 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))+$
$y_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_3 * cos(bt) * cos(gm) + z - z_1);$
resv[6][0] = $(x_2 - x_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_4 * cos(bt) * sin(gm) + y - y_1);$
resv[7][0] = $(x_2 - x_1) * (x_4 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))+$
$y_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_4 * cos(bt) * cos(gm) + z - z_1);$
resv[8][0] = $gm;$
break;

case 2: /*** *Parallel to Y axis* ***/
jac[0][0] = $cos(al) * cos(bt);$ /*** *Jacobian Matrix* ***/
jac[0][1] = $-sin(al) * sin(bt);$
jac[0][2] = 0.0;
jac[0][3] = 0.0;
jac[0][4] = 0.0;
jac[0][5] = 0.0;

jac[1][0] = $cos(al) * sin(bt) * sin(gm) + sin(al) * cos(gm);$
jac[1][1] = $sin(al) * cos(bt) * sin(gm);$
jac[1][2] = $sin(al) * sin(bt) * cos(gm) + cos(al) * sin(gm);$
jac[1][3] = 0.0;
jac[1][4] = 0.0;
jac[1][5] = 0.0;

jac[2][0] = $cos(al) * sin(bt) * cos(gm) - sin(al) * sin(gm);$
jac[2][1] = $sin(al) * cos(bt) * cos(gm);$
jac[2][2] = $cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm);$
jac[2][3] = 0.0;
jac[2][4] = 0.0;
jac[2][5] = 0.0;

jac[3][0] = $-(x_3 * sin(al) * sin(al) * cos(bt) * cos(bt))+$
$(x_3 * cos(al) * cos(al) * cos(bt) * cos(bt))-$

$(2.0 * y_3 * sin(al) * cos(al) * cos(bt) * cos(bt)) + (z_3 * cos(al) * sin(bt) * cos(bt)) -$
$(x_1 * cos(al) * cos(bt)) + (2.0 * x_3 * sin(al) * cos(al) * sin(bt) * sin(gm) * cos(gm)) +$
$(x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(x_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(2.0 * y_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(z_3 * cos(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm)) - (y_1 * cos(al) * sin(bt) * sin(gm)) -$
$(x_3 * cos(al) * cos(al) * cos(gm) * cos(gm)) + (x_3 * sin(al) * sin(al) * cos(gm) * cos(gm)) +$
$(2.0 * x_3 * cos(al) * sin(al) * cos(gm) * sin(bt) * sin(gm)) +$
$(2.0 * y_3 * cos(al) * sin(al) * cos(gm) * cos(gm)) -$
$(z_3 * sin(al) * cos(gm) * cos(bt) * sin(gm)) + (y_1 * sin(al) * cos(gm)) -$
$(x_3 * cos(al) * cos(al) * sin(gm) * sin(gm)) + (x_3 * sin(al) * sin(al) * sin(gm) * sin(gm)) -$
$(2.0 * x_3 * cos(al) * sin(al) * sin(gm) * sin(bt) * cos(gm)) +$
$(2.0 * y_3 * cos(al) * sin(al) * sin(gm) * sin(gm)) +$
$(z_3 * sin(al) * sin(gm) * cos(bt) * cos(gm)) - (z_1 * sin(al) * sin(gm)) -$
$(2.0 * x_3 * sin(al) * cos(al) * sin(gm) * sin(bt) * cos(gm)) +$
$(x_3 * cos(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) -$
$(x_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) -$
$(2.0 * y_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) -$
$(z_3 * cos(bt) * cos(al) * sin(bt) * cos(gm) * cos(gm)) +$
$(z_1 * cos(al) * sin(bt) * cos(gm)) + (x * cos(al) * cos(bt)) + (y * cos(al) * sin(bt) * sin(gm)) +$
$(y * sin(al) * cos(gm)) + (z * sin(al) * sin(gm)) - (z * cos(al) * sin(bt) * cos(gm));$

jac[3][1] $= -(2.0 * x_3 * cos(al) * sin(bt) * cos(bt) * sin(bt)) +$
$(2.0 * y_3 * sin(al) * sin(al) * cos(bt) * sin(bt)) + (z_3 * sin(al) * cos(bt) * cos(bt)) -$
$(z_3 * sin(al) * sin(bt) * sin(bt)) + (x_1 * sin(al) * sin(bt)) +$
$(x_3 * sin(al) * sin(al) * cos(bt) * sin(gm) * cos(gm)) +$
$(x_3 * sin(al) * sin(al) * cos(bt) * sin(gm) * cos(gm)) +$
$(2.0 * x_3 * sin(al) * cos(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm)) -$
$(2.0 * y_3 * sin(al) * sin(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm)) -$
$(z_3 * sin(al) * cos(bt) * cos(bt) * sin(gm) * sin(gm)) +$
$(z_3 * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) - (y_1 * sin(al) * cos(bt) * sin(gm)) -$
$(x_3 * cos(al) * cos(al) * cos(gm) * cos(bt) * sin(gm)) -$
$(z_3 * cos(al) * cos(gm) * sin(bt) * sin(gm)) +$
$(x_3 * sin(al) * sin(al) * sin(gm) * cos(bt) * cos(gm)) +$
$(z_3 * cos(al) * sin(gm) * cos(gm) * sin(bt)) -$
$(x_3 * sin(al) * sin(al) * sin(gm) * cos(bt) * cos(gm)) +$
$(2.0 * x_3 * sin(al) * cos(al) * sin(bt) * cos(bt) * cos(gm) * cos(gm)) -$
$(2.0 * y_3 * sin(al) * sin(al) * sin(bt) * cos(bt) * cos(gm) * cos(gm)) +$

$(z_3 * cos(bt) * cos(bt) * sin(al) * cos(gm) * cos(gm)) + (z_1 * sin(al) * cos(bt) * cos(gm)) -$
$(x * sin(al) * sin(bt)) + (y * sin(al) * cos(bt) * sin(gm)) - (z * sin(al) * cos(bt) * cos(gm));$

jac[3][2] $= (x_3 * sin(al) * sin(al) * sin(bt) * cos(gm) * cos(gm)) -$
$(x_3 * sin(al) * sin(al) * sin(bt) * sin(gm) * sin(gm)) +$
$(2.0 * x_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * cos(gm)) -$
$(2.0 * y_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * cos(gm)) -$
$(2.0 * z_3 * sin(al) * sin(bt) * cos(bt) * sin(gm) * cos(gm)) - (y_1 * sin(al) * sin(bt) * cos(gm)) +$
$(2.0 * x_3 * sin(al) * cos(al) * cos(gm) * sin(gm)) -$
$(x_3 * cos(al) * cos(al) * cos(gm) * cos(gm) * sin(bt)) +$
$(x_3 * cos(al) * cos(al) * sin(gm) * sin(gm) * sin(bt)) +$
$(y_3 * cos(al) * cos(al) * cos(gm) * sin(gm)) + (z_3 * cos(al) * cos(gm) * cos(gm) * cos(bt)) -$
$(z_3 * cos(al) * cos(bt) * sin(gm) * sin(gm)) + (y_1 * cos(al) * sin(gm)) -$
$(x_3 * sin(al) * cos(al) * sin(gm) * cos(gm)) +$
$(x_3 * cos(al) * cos(al) * cos(gm) * cos(gm) * cos(bt)) -$
$(x_3 * cos(al) * cos(al) * sin(gm) * sin(gm) * sin(bt)) -$
$(2.0 * y_3 * cos(al) * cos(al) * sin(gm) * cos(gm)) -$
$(z_3 * cos(al) * cos(gm) * cos(gm) * cos(bt)) +$
$(z_3 * cos(al) * sin(gm) * sin(gm) * cos(bt)) + (z_1 * cos(al) * cos(gm)) -$
$(x_3 * sin(al) * sin(al) * cos(gm) * cos(gm) * sin(bt)) +$
$(x_3 * sin(al) * sin(al) * sin(gm) * sin(gm) * sin(bt)) -$
$(2.0 * x_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * sin(gm)) +$
$(2.0 * y_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * sin(gm)) +$
$(2.0 * z_3 * cos(bt) * sin(al) * sin(bt) * cos(gm) * sin(gm)) -$
$(z_1 * sin(al) * sin(bt) * sin(gm)) + (y * sin(al) * sin(bt) * cos(gm)) +$
$(y * cos(al) * sin(gm)) - (z * cos(al) * cos(gm)) + (z * sin(al) * sin(bt) * sin(gm));$

jac[3][3] $= sin(al) * cos(bt);$
jac[3][4] $= sin(al) * sin(bt) * sin(gm) - cos(al) * cos(gm);$
jac[3][5] $= -cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm);$

jac[4][0] $= (y_2 - y_1) * (-x3 * sin(al) * cos(bt) - y3 * cos(al) * cos(bt));$
jac[4][1] $= (y_2 - y_1) * (-x3 * cos(al) * sin(bt) + y3 * sin(al) * sin(bt) + z3 * cos(bt));$
jac[4][2] $= 0.0;$
jac[4][3] $= (y_2 - y_1);$
jac[4][4] $= 0.0;$
jac[4][5] $= 0.0;$

$$jac[5][0] = (y_2 - y_1) * (x_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))+$$
$$y_3 * (cos(al) * sin(bt) * cos(gm) - sin(al) * sin(gm)));$$
$$jac[5][1] = (y_2 - y_1) * (x_3 * (-cos(al) * cos(bt) * cos(gm)) + y_3 * (sin(al)*$$
$$cos(bt) * cos(gm)) - z_3 * sin(bt) * cos(gm));$$
$$jac[5][2] = (y_2 - y_1) * (x_3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+$$
$$y_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_3 * cos(bt) * sin(gm));$$
$$jac[5][3] = 0.0;$$
$$jac[5][4] = 0.0;$$
$$jac[5][5] = (y_2 - y_1);$$

$$jac[6][0] = (y_2 - y_1) * (-x_4 * sin(al) * cos(bt) - y_4 * cos(al) * cos(bt));$$
$$jac[6][1] = (y_2 - y_1) * (-x_4 * cos(al) * sin(bt) + y_4 * sin(al) * sin(bt) + z_4 * cos(bt));$$
$$jac[6][2] = 0.0;$$
$$jac[6][3] = (y_2 - y_1);$$
$$jac[6][4] = 0.0;$$
$$jac[6][5] = 0.0;$$

$$jac[7][0] = (y_2 - y_1) * (x_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))+$$
$$y_4 * (cos(al) * sin(bt) * cos(gm) - sin(al) * sin(gm)));$$
$$jac[7][1] = (y_2 - y_1) * (x4 * (-cos(al) * cos(bt) * cos(gm)) + y_4 * (sin(al)*$$
$$cos(bt) * cos(gm)) - z_4 * sin(bt) * cos(gm));$$
$$jac[7][2] = (y_2 - y_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+$$
$$y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_4 * cos(bt) * sin(gm));$$
$$jac[7][3] = 0.0;$$
$$jac[7][4] = 0.0;$$
$$jac[7][5] = (y_2 - y_1);$$

$$jac[8][0] = 0.0;$$
$$jac[8][1] = 1.0;$$
$$jac[8][2] = 0.0;$$
$$jac[8][3] = 0.0;$$
$$jac[8][4] = 0.0;$$
$$jac[8][5] = 0.0;$$

/* * * *Residual vector* * * */

$$resv[0][0] = sin(al) * cos(bt);$$
$$resv[1][0] = ((sin(al) * sin(bt) * sin(gm)) - (cos(al) * cos(gm)) + 1.0);$$
$$resv[2][0] = ((cos(al) * sin(gm)) + (sin(al) * sin(bt) * cos(gm)));$$

$\text{resv}[3][0] = ((x_3 * cos(al) * sin(al) * cos(bt) * cos(bt))-$
$(y_3 * sin(al) * sin(al) * cos(bt) * cos(bt)) + (z_3 * sin(al) * sin(bt) * cos(bt))-$
$(x_1 * sin(al) * cos(bt)) + (x_3 * sin(al) * sin(al) * sin(bt) * sin(gm) * cos(gm))+$
$(x_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm))-$
$(y_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm))-$
$(z_3 * sin(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm)) - (y_1 * sin(al) * sin(bt) * sin(gm))-$
$(x_3 * sin(al) * cos(al) * cos(gm) * cos(gm))-$
$(x_3 * cos(al) * cos(al) * cos(gm) * sin(bt) * sin(gm))-$
$(y_3 * cos(al) * cos(al) * cos(gm) * cos(gm))+$
$(z_3 * cos(al) * cos(gm) * cos(bt) * sin(gm))+$
$(y_1 * cos(al) * cos(gm)) - (x_3 * sin(al) * cos(al) * sin(gm) * sin(gm))+$
$(x_3 * cos(al) * cos(al) * sin(gm) * sin(bt) * cos(gm))-$
$(y_3 * cos(al) * cos(al) * sin(gm) * sin(gm)) + (z_1 * cos(al) * sin(gm))-$
$(x_3 * sin(al) * sin(al) * sin(gm) * sin(bt) * cos(gm))+$
$(x_3 * sin(al) * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))-$
$(y_3 * sin(al) * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm))-$
$(z_3 * cos(bt) * sin(al) * sin(bt) * cos(gm) * cos(gm)) + (z_1 * sin(al) * sin(bt) * cos(gm))+$
$(x * sin(al) * cos(bt)) + (y * sin(al) * sin(bt) * sin(gm)) - (y * cos(al) * cos(gm))-$
$(z * cos(al) * sin(gm)) - (z * sin(al) * sin(bt) * cos(gm)));$
$\text{resv}[4][0] = (y_2 - y_1) * (x_3 * cos(al) * cos(bt) - y_3 * sin(al) * cos(bt) + z_3 * sin(bt) + x - x_1);$
$\text{resv}[5][0] = (y_2 - y_1) * (x_3 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))+$
$y_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_3 * cos(bt) * cos(gm) + z - z_1);$
$\text{resv}[6][0] = (y_2 - y_1) * (x_4 * cos(al) * cos(bt) - y_4 * sin(al) * cos(bt) + z_4 * sin(bt) + x - x_1);$
$\text{resv}[7][0] = (y_2 - y_1) * (x4 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))+$
$y_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_4 * cos(bt) * cos(gm) + z - z_1);$
$\text{resv}[8][0] = bt;$
break;

case 3: /*** *Parallel to Z axis* ***/
jac[0][0] = 0.0; /*** *Jacobian Matrix* ***/
jac[0][1] = $cos(bt)$;
jac[0][2] = 0.0;
jac[0][3] = 0.0;
jac[0][4] = 0.0;
jac[0][5] = 0.0;

jac[1][0] = 0.0;
jac[1][1] = $-sin(bt) * sin(gm)$;
jac[1][2] = $cos(bt) * cos(gm)$;

$\mathrm{jac}[1][3] = 0.0;$
$\mathrm{jac}[1][4] = 0.0;$
$\mathrm{jac}[1][5] = 0.0;$

$\mathrm{jac}[2][0] = 0.0;$
$\mathrm{jac}[2][1] = -sin(bt) * cos(gm);$
$\mathrm{jac}[2][2] = -cos(bt) * sin(gm);$
$\mathrm{jac}[2][3] = 0.0;$
$\mathrm{jac}[2][4] = 0.0;$
$\mathrm{jac}[2][5] = 0.0;$

$\mathrm{jac}[3][0] = (x_3 * sin(al) * cos(bt) * sin(bt)) + (y_3 * cos(al) * cos(bt) * sin(bt)) -$
$(x_3 * sin(al) * sin(bt) * sin(gm) * sin(gm) * cos(bt)) -$
$(y_3 * cos(al) * sin(bt) * cos(bt) * sin(gm) * sin(gm)) -$
$(x_3 * sin(al) * sin(bt) * cos(gm) * cos(gm) * cos(bt)) -$
$(y_3 * cos(al) * sin(bt) * cos(gm) * cos(gm) * cos(bt));$

$\mathrm{jac}[3][1] = (x_3 * cos(al) * sin(bt) * sin(bt)) - (x_3 * cos(al) * cos(bt) * cos(bt)) +$
$(y_3 * sin(al) * cos(bt) * cos(bt)) - (y_3 * sin(al) * sin(bt) * sin(bt)) -$
$(2.0 * z_3 * sin(bt) * cos(bt)) - (x * cos(bt)) + (x_1 * cos(bt)) +$
$(x_3 * cos(al) * cos(bt) * cos(bt) * sin(gm) * sin(gm)) -$
$(x_3 * cos(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(y_3 * sin(al) * cos(bt) * cos(bt) * sin(gm) * sin(gm)) +$
$(y_3 * sin(al) * sin(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(2.0 * z_3 * cos(bt) * sin(bt) * sin(gm) * sin(gm)) -$
$(y * sin(bt) * sin(gm)) + (y_1 * sin(bt) * sin(gm)) +$
$(x_3 * cos(al) * cos(bt) * cos(bt) * cos(gm) * cos(gm)) -$
$(x_3 * cos(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) -$
$(y_3 * sin(al) * cos(bt) * cos(bt) * cos(gm) * cos(gm)) +$
$(y_3 * sin(al) * sin(bt) * sin(bt) * cos(gm) * cos(gm)) +$
$(2.0 * z_3 * cos(gm) * cos(gm) * cos(bt) * sin(bt)) + (z * sin(bt) * cos(gm)) -$
$(z_1 * sin(bt) * cos(gm));$

$\mathrm{jac}[3][2] = (2.0 * y_3 * sin(al) * sin(bt) * cos(bt) * sin(gm) * cos(gm)) +$
$(2.0 * z_3 * cos(bt) * cos(bt) * sin(gm) * cos(gm)) +$
$(y * cos(bt) * cos(gm)) - (y_1 * cos(bt) * cos(gm)) +$
$(2.0 * y_3 * sin(al) * sin(bt) * cos(bt) * cos(gm) * sin(gm)) +$
$(2.0 * z_3 * cos(bt) * cos(bt) * cos(gm) * sin(gm)) +$

$(z * cos(bt) * sin(gm)) - (z_1 * cos(bt) * sin(gm));$

$jac[3][3] = -sin(bt);$
$jac[3][4] = cos(bt) * sin(gm);$
$jac[3][5] = -cos(bt) * cos(gm);$

$jac[4][0] = (z_2 - z_1) * (-x_3 * sin(al) * cos(bt) - y_3 * cos(al) * cos(bt));$
$jac[4][1] = (z_2 - z_1) * (-x_3 * cos(al) * sin(bt) + y_3 * sin(al) * sin(bt) + z_3 * cos(bt));$
$jac[4][2] = 0.0;$
$jac[4][3] = (z_2 - z_1);$
$jac[4][4] = 0.0;$
$jac[4][5] = 0.0;$

$jac[5][0] = (z_2 - z_1) * (x_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) +$
$y_3 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)));$
$jac[5][1] = (z_2 - z_1) * (x_3 * (cos(al) * cos(bt) * sin(gm)) +$
$y_3 * (-sin(al) * cos(bt) * sin(gm)) - z_3 * sin(bt) * sin(gm));$
$jac[5][2] = (z_2 - z_1) * (x_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)) +$
$y_3 * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm)) + z_3 * cos(bt) * cos(gm));$
$jac[5][3] = 0.0;$
$jac[5][4] = (z_2 - z_1);$
$jac[5][5] = 0.0;$

$jac[6][0] = (z_2 - z_1) * (-x_4 * sin(al) * cos(bt) - y_4 * cos(al) * cos(bt));$
$jac[6][1] = (z_2 - z_1) * (-x_4 * cos(al) * sin(bt) + y_4 * sin(al) * sin(bt) + z_4 * cos(bt));$
$jac[6][2] = 0.0;$
$jac[6][3] = (z_2 - z_1);$
$jac[6][4] = 0.0;$
$jac[6][5] = 0.0;$

$jac[7][0] = (z_2 - z_1) * (x_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) +$
$y_4 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)));$
$jac[7][1] = (z_2 - z_1) * (x_4 * (cos(al) * cos(bt) * sin(gm)) +$
$y_4 * (-sin(al) * cos(bt) * sin(gm)) - z_4 * sin(bt) * sin(gm));$
$jac[7][2] = (z_2 - z_1) * (x_4 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)) +$
$y_4 * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm)) + z_4 * cos(bt) * cos(gm));$
$jac[7][3] = 0.0;$
$jac[7][4] = (z_2 - z_1);$

```
        jac[7][5] = 0.0;

        jac[8][0] = 1.0;
        jac[8][1] = 0.0;
        jac[8][2] = 0.0;
        jac[8][3] = 0.0;
        jac[8][4] = 0.0;
        jac[8][5] = 0.0;

        /*** residual vector ***/
        resv[0][0] = sin(bt);
        resv[1][0] = cos(bt) * sin(gm);
        resv[2][0] = (cos(bt) * cos(gm) - 1);
        resv[3][0] = (-(x3 * cos(al) * cos(bt) * sin(bt)) + (y3 * sin(al) * cos(bt) * sin(bt)) -
        (z3 * sin(bt) * sin(bt)) - (x * sin(bt)) + (x1 * sin(bt))+
        (x3 * cos(al) * sin(bt) * sin(gm) * cos(bt) * sin(gm))-
        (y3 * sin(al) * sin(bt) * sin(gm) * sin(gm) * cos(bt))+
        (z3 * cos(bt) * cos(bt) * sin(gm) * sin(gm)) + (y * cos(bt) * sin(gm))-
        (y1 * cos(bt) * sin(gm))-
        (x3 * cos(al) * sin(bt) * cos(gm) * cos(gm) * cos(bt))-
        (y3 * sin(al) * sin(bt) * cos(gm) * cos(gm) * cos(bt))-
        (z3 * cos(bt) * cos(bt) * cos(gm) * cos(gm)) - (z * cos(bt) * cos(gm))+
        (z1 * cos(bt) * cos(gm)));
        resv[4][0] = (z2-z1)*(x3*cos(al)*cos(bt)-y3*sin(al)*cos(bt)+z3*sin(bt)+x-x1);
        resv[5][0] = (z2 - z1) * (x3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+
        y3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) + z3 * cos(bt) * sin(gm) + y - y1);
        resv[6][0] = (z2-z1)*(x4*cos(al)*cos(bt)-y4*sin(al)*cos(bt)+z4*sin(bt)+x-x1);
        resv[7][0] = (z2 - z1) * (x4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))+
        y4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) + z4 * cos(bt) * sin(gm) + y - y1);
        resv[8][0] = al;
        break;
}

/*** Least Square Approach ***/
        for(i = 0; i < 6; i++)
        for(j = 0; j < 9; j++)
        trjac[i][j] = jac[j][i];
```

```
/*** Calling Functions ***/
matmul(6,9,6,&trjac[0][0],&jac[0][0],&protrjac[0][0]);
matinv(6,6,&protrjac[0][0],&invprotrjac[0][0]);
matmul(6,6,9,&invprotrjac[0][0],&trjac[0][0],&slpro[0][0]);
matmul(6,9,1,&slpro[0][0],&resv[0][0],&fpro[0][0]);

if((x1 > 0&&x3 > 0)&&(y1 > 0&&y3 > 0)&&(z1 > 0&&z3 > 0))
{
*tx = fpro[3][0]; *ty = fpro[4][0]; *tz = fpro[5][0];
}
else
{
*tx = -fpro[3][0]; *ty = -fpro[4][0]; *tz = -fpro[5][0];
}
}
/***Algorithm for the case where the component is inclined to two of the planes and
parallel to one plane***/
GLfloat xpjacnresv( GLint s, GLfloat *p11, GLfloat *p22, GLfloat *p33, GLfloat *p44,
                    GLfloat *tx, GLfloat *ty, GLfloat *tz )
{
GLfloat nbx,nby,nbz,nrx,nry,nrz;

/***Normals of the two inclined mating faces***/
nrx = cos(15*m);        nbx = -cos(15*m);
nry = -sin(15*m);       nby = sin(15*m);
nrz = 0.0;              nbz = 0.0;

for(i = 0; i < 3; i++)
{
p1[i] = *p11; *p11++;
p2[i] = *p22; *p22++;
p3[i] = *p33; *p33++;
p4[i] = *p44; *p44++;
}

x1 = p1[0]; y1 = p1[1]; z1 = p1[2];
x2 = p2[0]; y2 = p2[1]; z2 = p2[2];
if(s==1)
```

{

$x_3 = p_3[0]; y_3 = p_3[1]; z_3 = p_3[2];$

$x_4 = p_4[0]; y_4 = p_4[1]; z_4 = p_4[2];$

}

else

{

$x_3 = p_4[0]; y_3 = p_4[1]; z_3 = p_4[2];$

$x_4 = p_3[0]; y_4 = p_3[1]; z_4 = p_3[2];$

}

$x = 0.0; y = 0.0; z = 0.0;$

$gamma = 0.0; beta = 0.0; alpha = 0.0;$

$gm = (PI * gamma)/180.0;$

$bt = (PI * beta)/180.0;$

$al = (PI * alpha)/180.0;$

/* * **Jacobian Matrix** * */

$\text{jac}[0][0] = -(nrx) * sin(al) * cos(bt) - (nry) * cos(al) * cos(bt);$

$\text{jac}[0][1] = -(nrx) * cos(al) * sin(bt) + (nry) * sin(al) * sin(bt);$

$\text{jac}[0][2] = 0.0;$

$\text{jac}[0][3] = 0.0;$

$\text{jac}[0][4] = 0.0;$

$\text{jac}[0][5] = 0.0;$

$\text{jac}[1][0] = (nrx) * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) +$

$(nry) * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm));$

$\text{jac}[1][1] = (nrx) * (cos(al) * cos(bt) * sin(gm)) + (nry) * (-sin(al) * cos(bt) * sin(gm));$

$\text{jac}[1][2] = (nrx) * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)) +$

$(nry) * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm));$

$\text{jac}[1][3] = 0.0;$

$\text{jac}[1][4] = 0.0;$

$\text{jac}[1][5] = 0.0;$

$\text{jac}[2][0] = (nrx) * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) +$

$(nry) * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm));$

$\text{jac}[2][1] = (nrx) * (cos(al) * cos(bt) * cos(gm)) + (nry) * (sin(al) * cos(bt) * cos(gm));$

$\text{jac}[2][2] = (nrx) * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm)) +$

$(nry) * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm));$
jac[2][3] = 0.0;
jac[2][4] = 0.0;
jac[2][5] = 0.0;

jac[3][0] $= -y_3 * (nrx) * cos(al) * cos(al) * cos(bt) * cos(bt) -$
$x_3 * (nry) * cos(al) * cos(al) * cos(bt) * cos(bt) - x_1 * (nry) * cos(al) * cos(bt) +$
$y_3 * (nrx) * cos(al) * cos(al) * cos(gm) * cos(gm) - y_1 * (nrx) * cos(al) * cos(gm) +$
$x_3 * (nry) * cos(al) * cos(al) * cos(gm) * cos(gm);$
jac[3][1] $= z_3 * (nrx) * cos(al) * cos(bt) * cos(bt) -$
$z_3 * (nrx) * cos(bt) * cos(bt) * cos(al) * cos(gm) * cos(gm) +$
$z_1 * (nrx) * cos(al) * cos(bt) * cos(gm);$
jac[3][2] $= -z_3 * (nry) * cos(bt) * cos(gm) * cos(gm) * cos(al) +$
$z_3 * (nry) * cos(bt) * cos(gm) * cos(gm) * cos(al) - z_1 * (nry) * cos(al) * cos(gm);$
jac[3][3] $= (nrx) * cos(al) * cos(bt) + (nry) * sin(al) * cos(bt);$
jac[3][4] $= (nrx) * sin(al) * cos(gm) + (nrx) * cos(al) * sin(bt) * sin(gm) +$
$(nry) * cos(al) * cos(gm) - (nry) * sin(al) * sin(bt) * sin(gm);$
jac[3][5] $= (nrx) * sin(al) * sin(gm) - (nrx) * cos(al) * sin(bt) * cos(gm) +$
$(nry) * cos(al) * sin(gm) + (nry) * sin(al) * sin(bt) * cos(gm);$

jac[4][0] $= (x_2 - x_1) * (x_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))$
$+y_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$
jac[4][1] $= (x_2 - x_1) * (x_3 * (-cos(al) * cos(bt) * cos(gm))$
$+y_3 * (sin(al) * cos(bt) * cos(gm)) - z_3 * sin(bt) * cos(gm));$
jac[4][2] $= (x_2 - x_1) * (x_3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+y_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_3 * cos(bt) * sin(gm));$
jac[4][3] = 0.0;
jac[4][4] = 0.0;
jac[4][5] $= (x_2 - x_1);$

jac[5][0] $= (y_2 - y_1) * (x_3 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))$
$+y_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$
jac[5][1] $= (y_2 - y_1) * (x_3 * (-cos(al) * cos(bt) * cos(gm))$
$+y_3 * (sin(al) * cos(bt) * cos(gm)) - z_3 * sin(bt) * cos(gm));$
jac[5][2] $= (y_2 - y_1) * (x_3 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+y_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_3 * cos(bt) * sin(gm));$
jac[5][3] = 0.0;
jac[5][4] = 0.0;

$jac[5][5] = (y_2 - y_1);$

$jac[6][0] = (x_2 - x_1) * (x_3 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm))$
$+y_3 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)))$
$-(y_2 - y_1) * (-x_3 * sin(al) * cos(bt) - y3 * cos(al) * cos(bt));$
$jac[6][1] = (x_2-x_1)*(x_3*(cos(al)*cos(bt)*sin(gm))+y_3*(-sin(al)*cos(bt)*sin(gm))$
$+z_3 * sin(bt) * sin(gm)) - (y_2 - y_1) * (-x_3 * cos(al) * sin(bt)+$
$y_3 * sin(al) * sin(bt) + z_3 * cos(bt));$
$jac[6][2] = (x_2 - x_1) * (x_3 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm))$
$+y_3 * (-cos(al) * sin(gm) - sin(al) * sin(bt) * cos(gm)) - z_3 * cos(bt) * cos(gm));$
$jac[6][3] = (y_2 - y_1);$
$jac[6][4] = (x_2 - x_1);$
$jac[6][5] = 0.0;$

$jac[7][0] = (x_2 - x_1) * (x_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))$
$+y_4 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$
$jac[7][1] = (x_2 - x_1) * (x_4 * (-cos(al) * cos(bt) * cos(gm))$
$+y_4 * (sin(al) * cos(bt) * cos(gm)) - z_4 * sin(bt) * cos(gm));$
$jac[7][2] = (x_2 - x_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_4 * cos(bt) * sin(gm));$
$jac[7][3] = 0.0;$
$jac[7][4] = 0.0;$
$jac[7][5] = (x_2 - x_1);$

$jac[8][0] = (y_2 - y_1) * (x_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm))$
$+y_4 * (-sin(al) * sin(gm) + cos(al) * sin(bt) * cos(gm)));$
$jac[8][1] = (y_2 - y_1) * (x_4 * (-cos(al) * cos(bt) * cos(gm))$
$+y_4 * (sin(al) * cos(bt) * cos(gm)) - z_4 * sin(bt) * cos(gm));$
$jac[8][2] = (y_2 - y_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) - z_4 * cos(bt) * sin(gm));$
$jac[8][3] = 0.0;$
$jac[8][4] = 0.0;$
$jac[8][5] = (y_2 - y_1);$

$jac[9][0] = (x_2 - x_1) * (x_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm))$
$+y_4 * (-sin(al) * cos(gm) - cos(al) * sin(bt) * sin(gm)))-$
$(y_2 - y_1) * (-x_4 * sin(al) * cos(bt) - y_4 * cos(al) * cos(bt));$
$jac[9][1] = (x_2 - x_1) * (x_4 * (cos(al) * cos(bt) * sin(gm))$

$+y_4*(-sin(al)*cos(bt)*sin(gm))-z_4*sin(bt)*sin(gm))-$
$(y_2-y_1)*(-x_4*cos(al)*sin(bt)-y_4*sin(al)*sin(bt)+z_4*cos(bt));$
jac[9][2] $=(x_2-x_1)*(x_4*(-sin(al)*sin(gm)+cos(al)*sin(bt)*cos(gm))$
$+y_4*(-cos(al)*sin(gm)-sin(al)*sin(bt)*cos(gm))+z_4*cos(bt)*cos(gm));$
jac[9][3] $=(y_2-y_1);$
jac[9][4] $=(x_2-x_1);$
jac[9][5] = 0.0;

jac[10][0] = 0.0;
jac[10][1] = 0.0;
jac[10][2] = 1.0;
jac[10][3] = 0.0;
jac[10][4] = 0.0;
jac[10][5] = 0.0;

/****Residual Vector****/

resv[0][0] $=(nrx)*cos(al)*cos(bt)-(nry)*sin(al)*cos(bt)+nbx;$

resv[1][0] $=(nrx)*(sin(al)*cos(gm)+cos(al)*sin(bt)*sin(gm))+$
$(nry)*(cos(al)*cos(gm)-sin(al)*sin(bt)*sin(gm))+nby;$

resv[2][0] $=(nrx)*(sin(al)*sin(gm)-cos(al)*sin(bt)*cos(gm))+$
$(nry)*(cos(al)*sin(gm)+sin(al)*sin(bt)*cos(gm))+nbz;$

resv[3][0] $=x_3*(nrx)*cos(al)*cos(al)*cos(bt)*cos(bt)-$
$x_1*(nrx)*cos(al)*cos(bt)+$
$y_3*(nry)*cos(al)*cos(al)*cos(gm)*cos(gm)-y_1*(nry)*cos(al)*cos(gm);$

resv[4][0] $=(x_2-x_1)*(x_3*(sin(al)*sin(gm)-cos(al)*sin(bt)*cos(gm))$
$+y_3*(cos(al)*sin(gm)+sin(al)*sin(bt)*cos(gm))+z_3*cos(bt)*cos(gm)+z-z_1);$

resv[5][0] $=(y_2-y_1)*(x_3*(sin(al)*sin(gm)-cos(al)*sin(bt)*cos(gm))$
$+y_3*(cos(al)*sin(gm)+sin(al)*sin(bt)*cos(gm))+z_3*cos(bt)*cos(gm)+z-z_1);$

resv[6][0] $=(x_2-x_1)*(x_3*(sin(al)*cos(gm)+cos(al)*sin(bt)*sin(gm))$
$+y_3*(cos(al)*cos(gm)-sin(al)*sin(bt)*sin(gm))-z_3*cos(bt)*sin(gm)+y-y_1)-$
$(y_2-y_1)*(x3*cos(al)*cos(bt)-y_3*sin(al)*cos(bt)+z_3*sin(bt)+x-x_1);$

$\text{resv}[7][0] = (x_2 - x_1) * (x_4 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))$
$+ y_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_4 * cos(bt) * cos(gm) + z - z_1);$

$\text{resv}[8][0] = (y_2 - y_1) * (x_4 * (sin(al) * sin(gm) - cos(al) * sin(bt) * cos(gm))$
$+ y_4 * (cos(al) * sin(gm) + sin(al) * sin(bt) * cos(gm)) + z_4 * cos(bt) * cos(gm) + z - z_1);$

$\text{resv}[9][0] = (x_2 - x_1) * (x_4 * (sin(al) * cos(gm) + cos(al) * sin(bt) * sin(gm))$
$+ y_4 * (cos(al) * cos(gm) - sin(al) * sin(bt) * sin(gm)) + z_4 * cos(bt) * sin(gm) + y - y_1) -$
$(y_2 - y_1) * (x_4 * cos(al) * cos(bt) - y_4 * sin(al) * cos(bt) + z_4 * sin(bt) + x - x_1);$

$\text{resv}[10][0] = gm;$

$\text{for}(i = 0; i < 6; i++)$
$\text{for}(j = 0; j < 9; j++)$
$trjac[i][j] = jac[j][i];$

$\text{matmul}(6, 9, 6, \&trjac[0][0], \&jac[0][0], \&protrjac[0][0]);$
$\text{matinv}(6, 6, \&protrjac[0][0], \&invprotrjac[0][0]);$
$\text{matmul}(6, 6, 9, \&invprotrjac[0][0], \&trjac[0][0], \&slpro[0][0]);$
$\text{matmul}(6, 9, 1, \&slpro[0][0], \&resv[0][0], \&fpro[0][0]);$

$\text{if}((x_1 > 0 \&\& x_3 > 0) \&\& (y_1 > 0 \&\& y_3 > 0) \&\& (z_1 > 0 \&\& z_3 > 0))$
$\{$
$*t_x = fpro[3][0]; *t_y = fpro[4][0]; *t_z = fpro[5][0];$
$\}$
else
$\{$
$*t_x = -fpro[3][0]; *t_y = -fpro[4][0]; *t_z = -fpro[5][0];$
$\}$

$\}$

Appendix (D)
About Representation Scheme used (OpenGL)

In this work, OpenGL (Open Graphics Library) is used for rendering different components of an assembly. The algorithm for assembly procedure has been written in "C" language which has good interface with OpenGL. The representation scheme being used is Boundary representation (B-rep) as the mating conditions are related to the faces, edges and vertices of the assembled parts.

Why OpenGL ?

Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C^{++}, FORTRAN, Ada, and Java. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

Besides this, using OpenGL has the following advantages:

- **Most Widely Adopted Graphics Standard**

  OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions.

- **High Visual Quality and Performance**

  Any visual computing application requiring maximum performance-from 3D animation to CAD to visual simulation can exploit high-quality, high-performance OpenGL capabilities. These capabilities allow developers in diverse markets such as broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to produce and display incredibly compelling 2D and 3D graphics.

A133673

Date Slip

The book is to be returned on the date last stamped.

A133673